Facultad
de
Ciencias

DESARROLLO DE LA LÓGICA DE UNOS COMPONENTES SOFTWARE PARA LA PREDICCIÓN SOBRE SERIES DE TIEMPO

(Development of software components logic for time series prediction)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Jaime Céspedes Sisniega

Director: Cristina Tirnauca

Co-Director: Miguel Sierra Sánchez

Septiembre - 2017

Índice

Índice de figuras

Índice de tablas

Agradecimientos

En primer lugar, me gustaría agradecer a mi familia y amigos el apoyo recibido durante todos estos años de carrera.

También agradecer a CIC Consulting Informático por confiar en mí y darme la oportunidad de desarrollar el proyecto en su empresa, y más en concreto a los compañeros del departamento de IDbox por ayudarme con el desarrollo del mismo.

Por último, dar las gracias a Cristina por ayudarme desde un primer momento y darme las pautas necesarias para poder afrontar la realización de este trabajo de fin de grado.

Resumen

Hoy en día, con el avance en el ámbito de las tecnologías motivado en parte por la generación de grandes volúmenes de datos, son muchas las empresas que se plantean sacarles un rendimiento que les proporcione la información suficiente para una posterior toma de decisiones.

Desde el punto de vista de la industria, estos avances permiten hacer uso de los datos recopilados a través de distintas fuentes de origen. Varias de estas fuentes de información se pueden encontrar en sensores o controladores lógicos programables (PLC). Por lo tanto, la industria en su denominada versión 4.0 tiene por objetivo proporcionar las herramientas necesarias para automatizar parte de su proceso a través del intercambio de información entre los distintos dispositivos que lo conforman, ya sean sensores, máquinas, servidores o sistemas de control y monitorización.

En este trabajo se plantea desarrollar la lógica de una serie de componentes software para el producto de monitorización IDbox de la empresa CIC Consulting Informático, que sean capaces de realizar predicciones sobre series de tiempo. Para ello, el uso de algoritmos de aprendizaje automático facilita la generación de modelos matemáticos sobre los que obtener las predicciones.

Palabras clave: Aprendizaje automático, series de tiempo, redes neuronales, máquinas de soporte vectorial, industria 4.0

Abstract

Nowadays, with the progress in the technology field motivated in part by the generation of large volumes of data, many companies are considering using this data to extract valuable information for further decition making policies.

From an industry point of view, these advances allow to make use of the data collected from very different sources. Several of these information sources are represented by sensors or programmable logic controllers (PLCs). Therefore, the industry in its so-called version 4.0 aims to provide the necessary tools to automate part of its process through the information exchange between the different devices that make it up, such as sensors, machines, servers or control and monitoring systems.

This work aims to develop the logic of a series of software components for the IDbox monitoring product of the CIC Consulting Informático for time series forecasting. For this, Machine Learning algorithms are used to generate mathematical models to obtain predictions.

Keywords: Machine learning, time series, neural networks, support vector machines, industry 4.0

1. Introducción

En los últimos años, la evolución de la tecnología ha permitido el desarrollo de computadores con una mayor capacidad de cálculo, lo que se traduce en un menor tiempo a la hora de ejecutar programas o aplicaciones que requieren un alto coste computacional. Con este avance ha sido posible el desarrollo de algoritmos más robustos y eficientes para distintas tareas dentro del campo del aprendizaje automático como pueden ser la clasificación, regresión y detección de anomalías y patrones.

La industria 4.0 [1] es descrita como el surgimiento de una nueva era tecnológica en el ámbito de la industria. Esto hace posible que los datos proporcionados por las máquinas, sensores o sistemas de información sirvan de origen para la toma de decisiones. Multitud de empresas dedicadas a este sector son las que optan por implementar en sus productos herramientas que les permitan obtener un mayor redimiento de los datos que poseen. Esta generación continua de altos volúmenes de datos es conocida como *Big Data* [2]. Es en este punto en donde entran en escena los algoritmos de aprendizaje automático, los cuales permiten extraer el conocimiento necesario para la futura toma de decisiones en función de un entrenamiento realizado sobre los datos almacenados con anterioridad.

El objetivo principal del proyecto consiste en el desarrollo de la lógica de varios componentes software que permitan la predicción de los valores de una señal cuya variación se produce en el tiempo. Dicha predicción se realiza única y exclusivamente haciendo uso de algoritmos de aprendizaje automático, por lo que se basa en un entrenamiento previo sobre los datos de la señal para un periodo histórico de tiempo acotado. Los datos utilizados para el entrenamiento podrán pertenecer tanto a una única señal, como a la combinación de los valores de múltiples señales. Para que la realización del proyecto sea posible, se han establecido una serie de tareas a completar:

- Análisis de los algoritmos de aprendizaje automático adecuados para realizar las predicciones.
- Desarrollo de pruebas sobre los algoritmos seleccionados a través del uso de datos reales de señales proporcionados por la empresa CIC Consulting Informático.
- Integración de los algoritmos en el BPM (Business Process Management Engine) del producto IDbox de CIC Consulting Informático a través de la implementación de la lógica de los componentes software.

El producto IDbox es una herramienta que reúne un conjunto de componentes software orientados a la monitorización y supervisión de procesos energéticos. Permite integrar diferentes orígenes de información como puede ser un sensor, un PLC (Programmable Logic Controller), un registrador de datos o *datalogger*, una base de datos, etc. A su vez, los datos recogidos y ya almacenados son procesados, pudiendo generarse diferentes tipos de estadísticos o cálculos definidos por el usuario. Desde la

interfaz web proporcionada por IDbox es posible visualizar informes, gráficas históricas y en tiempo real de las señales, correlaciones, mapas de calor, *dashboards*, etc. Todo ello de forma conjunta permite al usuario obtener una herramienta adecuada para la toma de decisiones.

Figura 1: Producto IDbox

Por su parte, el BPM es un motor de cálculo ubicado en la etapa del análisis de los datos, el cual proporciona al usuario un espacio de trabajo mediante el uso de unos componentes software visualmente tratados como cajas. Establece un flujo de trabajo que permite aplicar filtros y distintos tipos de operaciones a las señales deseadas, todo ello a través de la unión de varios de estos componentes.

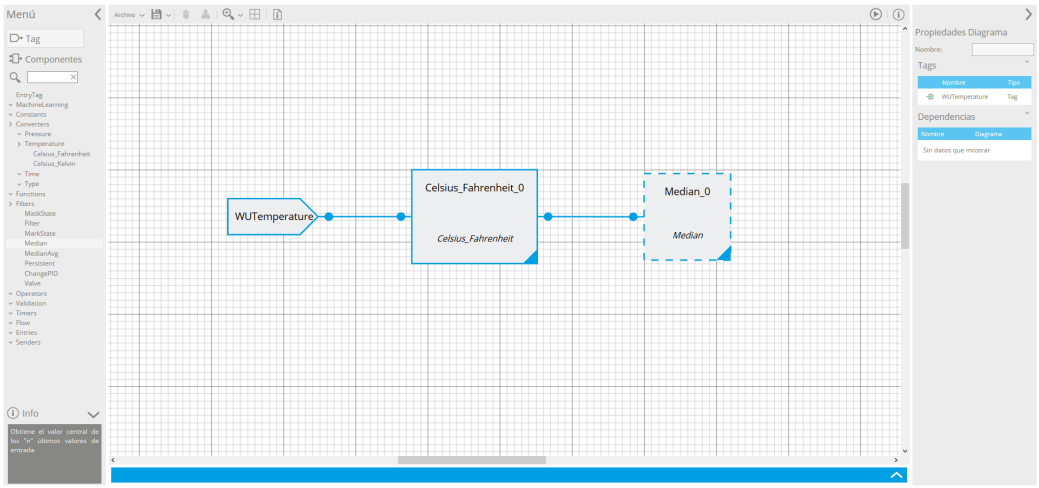

Figura 2: Flujo de trabajo en el BPM (Business Process Management)

La Figura 2 muestra un flujo de trabajo a modo de ejemplo en el que tomando una señal de temperatura se convierten sus valores de grados Celsius a grados Fahrenheit para posteriormente obtener la mediana como resultado final.

2. Tecnologías y herramientas

2.1. Lenguajes de programación

2.1.1. R

El lenguaje de programación R [3, 4] fue ideado principalmente para la computación estadística, el análisis de datos y su posterior visualización en forma de gráficos. Deriva del lenguaje S, el cual fue diseñado por John M. Chambers en los años 80 y usado por la comunidad estadística desde entonces. Una de las características que hacen destacar a R es la posibilidad que proporciona al usuario de ejecutar expresiones a través de su consola de manera simple e intuitiva. Además, proporciona multitud de paquetes desarrollados por la propia comunidad, los cuales pueden ser descargados incluso desde la consola.

2.1.2. C#

C# [5] es un lenguaje de programación orientado a objetos, desarrollado por un grupo de ingenieros de Microsoft e influenciado en gran parte por otros lenguajes como Java, C++ o Visual Basic. Como cualquier lenguaje de programación orientado a objetos, su funcionamiento principal se basa en el uso de clases, las cuales permiten definir nuevos tipos de objetos en función del tipo de problema que se desee tratar. Así mismo, es posible hacer uso de clases genéricas, interfaces, estructuras, acceder a los atributos correspondientes de cada objeto generado de forma sencilla. También posee más propiedades y otros componentes encontrados en la mayor parte de los lenguajes de programación orientados a objetos.

2.1.3. SQL

Uno de los lenguajes de programación más utilizado para la gestión de datos en sistemas de bases de datos relacionales es SQL (Structured Query Language) [6]. Desarrollado por IBM (International Business Machines) para realizar consultas, modificar y definir dichas bases de datos relacionales a través del uso de sentencias declarativas. Las operaciones principales de manipulación de datos son: INSERT (inserta valores), SELECT (devuelve una consulta), UPDATE (actualiza valores), DELETE (elimina valores).

2.1.4. R.NET

R.NET [7] es una librería que permite ejecutar código o scripts de R directamente desde C#. Haciendo uso de la función *Evaluate* que proporciona la librería, es posible pasar cualquier instrucción que se ejecutaría en la consola de R a través de un string, devolviendo un vector numérico o de caracteres con el que se puede trabajar en C#. Si por el contrario se desea ejecutar un script de R de forma completa sin tener que escribir instrucción a instrucción, la función *Source* implementa esta opción, facilitando la comprensión del código al separar de forma clara la parte escrita en C# de los scripts generados utilizando R.

2.2. Herramientas

2.2.1. RStudio

RStudio [8, 9] es un entorno de desarrollo integrado (IDE) que hace uso de la versión de R instalada. Dicho entorno de desarrollo incorpora un editor adaptado a R desde el cual se pueden abrir de forma simultanea múltiples ficheros mediante diferentes pestañas, lo que facilita el trabajo con múltiples archivos al mismo tiempo. Además, posee un apartado en el que se pueden visualizar todas las variables y funciones generadas, las cuales pueden ser creadas incluso durante el proceso de depuración.

2.2.2. Visual Studio

Visual Studio [10] es un entorno de desarrollo integrado (IDE) ideado por Microsoft que trata de facilitar al usuario la organización de su código a través de un explorador de soluciones con el fin de trabajar de forma más sencilla sobre los proyectos utilizados. Entre sus propiedades destaca la generación automática de código esqueleto al crear un nuevo proyecto, lo que permite al usuario ahorrarse el programar tareas repetitivas en el código. Del mismo modo, es posible personalizar multitud de aspectos a gusto del usuario y llevar a cabo tareas de refactorización sobre el código.

2.2.3. SQL Server Management Studio

SQL Server Management Studio [11] es una herramienta desarrollada por Microsoft que permite a través de su interfaz gráfica de usuario (GUI) gestionar bases de datos relacionales, creando y modificando componentes de la propia base de datos, así como tratar con temas relacionados con la seguridad de la misma.

3. Metodología

Si hablamos de predicción sobre series de tiempo estacionarias, los modelos ARMA (Modelos autorregresivos de media móvil) [12] se encuentran entre los más utilizados, mientras que para series de tiempo no estacionarias, los modelos ARIMA (Modelo autorregresivo integrado de media móvil) [13] pueden ser una opción válida. A su vez, existen modelos que combinan ARIMA con el uso de redes neuronales artificiales, descritas en la sección 3.3.1, dando lugar a modelos de carácter híbrido [14], los cuales consiguen mejorar ligeramente los resultados que se podrían obtener haciendo uso de ARIMA o de las redes neuronales de forma independiente. Una de las arquitecturas de redes neuronales más emergentes son las redes neuronales recurrentes [15], las cuales son usadas para conjuntos de datos dependientes de un orden o del tiempo, debido a que su arquitectura permite hacer la función de memoria temporal retroalimentando la capa oculta.

3.1. Planteamiento del problema

Por parte de CIC Consulting Informático se ha propuesto el uso exclusivo de algoritmos de aprendizaje automático que permitan obtener predicciones de los valores en función del tiempo. El problema se describe a continuación tanto para el uso de una señal como para el uso de múltiples señales distintas para la predicción de una única señal.

3.1.1. Predicción utilizando una señal

Para una única señal de entrenamiento se define S_i como un valor correspondiente a una señal en un momento determinado de tiempo, siendo $S_{i+1}, S_{i+2}, \ldots$ los valores que le siguen en el tiempo. Se toman los n valores anteriores al valor que se pretende predecir sobre un conjunto de datos formado por k muestras, quedando el planteamiento de la siguiente forma para el entrenamiento.

	Atributo 1	Atributo 2	...	Atributo n	Valor objetivo
Muestra 1	S_1	S_2	...	S_n	S_{n+1}
Muestra 2	S_2	S_3	...	S_{n+1}	S_{n+2}
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
Muestra k	S_k	S_{k+1}	...	S_{n+k-1}	S_{n+k}

Tabla 1: Entrenamiento con una señal

Una vez aplicado el algoritmo de aprendizaje automático deseado sobre el planteamiento descrito anteriormente, se genera un modelo matemático en base a las muestras utilizadas para el entrenamiento. Posteriormente se procede a realizar la predicción para conocer los p siguientes valores deseados de la señal. Para ello, se toman los n valores anteriores al primer valor que se desea predecir. A partir de la predicción del primer valor se pueden calcular los siguientes de la misma forma en que se entrena el modelo, es decir, retroalimentando los atributos necesarios con el valor predicho.

	Atributo 1	Atributo 2	...	Atributo n	Valor objetivo
Predicción 1	S_{1+k}	S_{2+k}	...	S_{n+k}	S_{n+1+k}
Predicción 2	S_{2+k}	S_{3+k}	...	S_{n+1+k}	S_{n+2+k}
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
Predicción p	S_{p+k}	S_{p+1+k}	...	$S_{n+p-1+k}$	S_{n+p+k}

Tabla 2: Predicción con una señal

3.1.2. Predicción utilizando múltiples señales

Para el entrenamiento sobre q señales distintas se define S_i^j como un valor correspondiente a la señal j en un momento i de tiempo, siendo $S_{i+1}^j, S_{i+2}^j, \ldots$ los valores que le siguen en el tiempo. La primera señal es establecida como la señal sobre la que se quiere obtener la predicción de los valores, mientras que para las señales restantes se obtendrán predicciones intermedias que permitan producir las predicciones de la señal objetivo, quedando el entrenamiento como se muestra a continuación en la Tabla 3.

	Señal 1				...	Señal q				Señal i
	Atr. 1	Atr. 2	...	Atr. n		Atr. 1	Atr. 2	...	Atr. n	Val. obj
Muest. 1	S_1^1	S_2^1	...	S_n^1		S_1^q	S_2^q	...	S_n^q	S_{n+1}^i
Muest. 2	S_2^1	S_3^1	...	S_{n+1}^1		S_2^q	S_3^q	...	S_{n+1}^q	S_{n+2}^i
$\vdots$	$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$		$\vdots$	$\vdots$
Muest. k	S_k^1	S_{k+1}^1	...	S_{n+k-1}^1		S_k^q	S_{k+1}^q	...	S_{n+k-1}^q	S_{n+k}^i

Tabla 3: Entrenamiento con varias señales

Una vez entrenados los q modelos distintos, uno por cada señal, se pueden predecir los p siguientes valores para la señal introducida como señal 1. Para ello, se han de predecir de forma simultanea los $p-1$ valores intermedios de las $q-1$ señales restantes. Esto es debido a que a partir del segundo valor que se quiera predecir, los valores utilizados para la predicción de la señal objetivo son fruto a su vez de predicciones previas, haciendo necesario obtener al menos las $p-1$ predicciones intermedias de las señales distintas de la objetivo. Sería posible de forma opcional, ya que no es el objetivo inicial del problema, obtener las predicciones completas de todas las señales utilizadas en el entrenamiento en vez de restringir la predicción a únicamente la primera señal, para ello habría que obtener los p valores de las señales distintas de la objetivo, en vez de los $p-1$ valores siguientes como está establecido actualmente.

	Señal 1				...	Señal q				Señal i
	Atr. 1	Atr. 2	...	Atr. n		Atr. 1	Atr. 2	...	Atr. n	Val. obj
Predic. 1	S^1_{1+k}	S^1_{2+k}	$\cdots$	S^1_{n+k}		S^q_{1+k}	S^q_{2+k}	$\cdots$	S^q_{n+k}	S^i_{n+1+k}
Predic. 2	S^1_{2+k}	S^1_{3+k}	$\cdots$	S^1_{n+1+k}		S^q_{2+k}	S^q_{3+k}	$\cdots$	S^q_{n+1+k}	S^i_{n+2+k}
$\vdots$	$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$		$\vdots$	$\vdots$
Predic. p	S^1_{p+k}	S^1_{p+1+k}	$\cdots$	$S^1_{n+p-1+k}$		S^q_{p+k}	S^q_{p+1+k}	$\cdots$	$S^q_{n+p-1+k}$	S^i_{n+p+k}

Tabla 4: Predicción con varias señales

3.2. Preprocesamiento de los datos

En ocasiones, con el fin de tener una convergencia más rápida de los algoritmos de aprendizaje automático o en situaciones en las cuales los diferentes atributos se encuentran en escalas de valores muy diferentes, es necesario normalizar o estandarizar los datos.

La normalización [16] permite obtener los valores en el rango $[0, 1]$ utilizando el valor máximo y mínimo del conjunto de datos.

$$\tilde{x} = \frac{x - min}{max - min}$$

Otra forma de preprocesar los datos es mediante la estandarización [16], obteniendo valores con $\mu = 0$ y $\sigma = 1$,

$$\tilde{x} = \frac{x - \mu}{\sigma}$$

siendo μ la media y σ la desviación estándar de los datos.

En el caso del problema planteado, se ha decidido utilizar la estandarización al proporcionar valores que no están acotados por un máximo y un mínimo como en el caso de la normalización. El principal problema que surge al existir unas cotas reside en que los datos de test han de estar contenidos en dicho rango de valores, de lo contrario las predicciones resultantes no tendrán valores razonables. De forma adicional, los valores atípicos o también denominados *outliers* pueden pasar desapercibidos si se normaliza el conjunto de datos, perdiendo información que puede ser relevante para el problema.

3.3. Algoritmos de aprendizaje automático

Con el objetivo de obtener unas predicciones lo más acertadas posibles para valores que varían en el tiempo, se hace uso de dos de los algoritmos más utilizados para tareas de clasificación, pero en este caso aplicados a la regresión: redes neuronales y máquinas de soporte vectorial. Ambos modelos se encuentran dentro de los algoritmos de aprendizaje supervisado [17], es decir, para cada muestra con que se realiza el entrenamiento existe una salida ya conocida. Algoritmos como la regresión

polinomial multivariante quedan descartados al ser necesario seleccionar un valor alto para el grado del polinomio usado para la predicción.

3.3.1. Redes neuronales artificiales

Las redes neuronales [18] pueden ser vistas como un modelo que imita la manera en que el cerebro humano es capaz de aprender. Fueron desarrolladas a lo largo de los años 80 y comienzos de los 90, a pesar de perder popularidad debido a su alto coste computacional para la época. Con el avance tecnológico de los últimos años y la correspondiente mejora a nivel de hardware de los computadores, las redes neuronales se han convertido en uno de los algoritmos de mayor uso para tareas de clasificación y regresión debido a su gran flexibilidad a la hora de adaptarse a los problemas cuyos datos no son linealmente separables.

El modelo simula el funcionamiento de las neuronas del cerebro y la manera en que se comunican unas con otras. Para ello, se toma un vector de entrada $x \in \mathbb{R}^n$ con los valores suministrados denominado capa de entrada, un número variable de capas ocultas con sus respectivas neuronas y una capa de salida con una o varias neuronas en caso de tratarse de un problema de clasificación con múltiples clases. Por ejemplo, la Figura 3 muestra la arquitectura de una red neuronal con una única capa oculta y un nodo de salida. En caso de existir múltiples capas ocultas, estas serán conectadas de la misma forma que se conecta la capa de entrada con la capa oculta en la Figura 3, es decir, cada uno de los nodos conectado a todos los nodos de la siguiente capa.

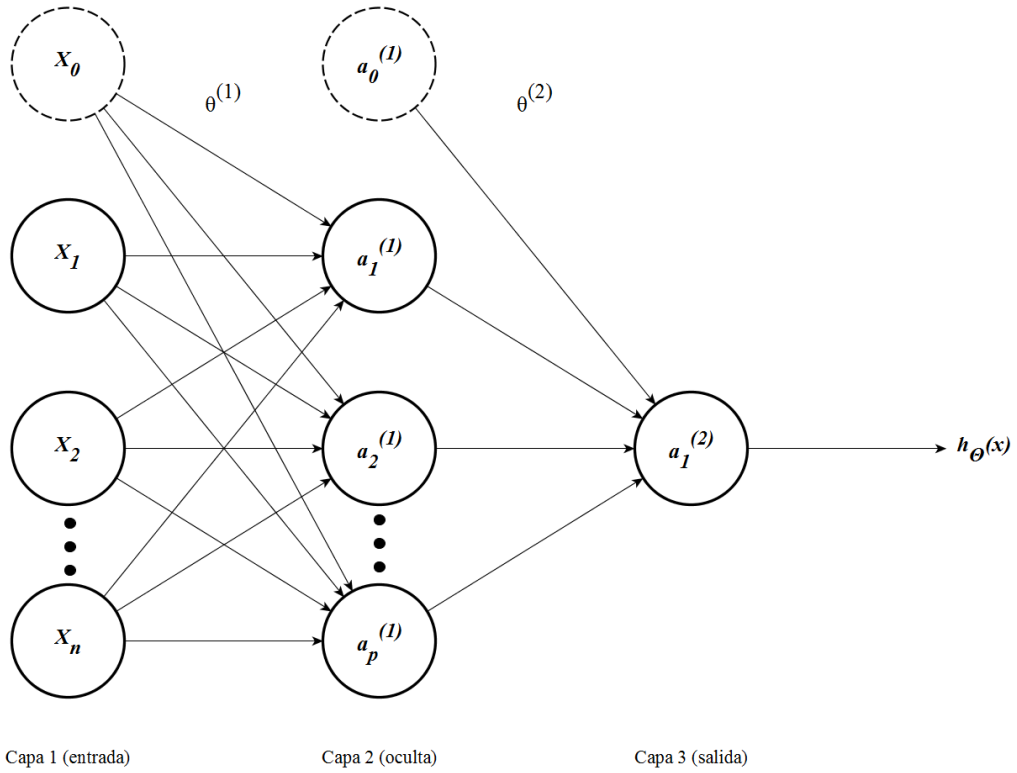

Figura 3: Red neuronal

Como el paquete de R utilizado, descrito en la sección 3.3.1.1, solo permite el uso de una capa oculta a continuación se explica el funcionamiento de una red neuronal con una sola capa oculta y una sola neurona de salida, tal y como se muestra en la Figura 3.

Cada una de las neuronas $a_i^{(1)}$ perteneciente a la capa oculta recibe un vector de pesos $\theta_i^{(1)} \in \mathbb{R}^{n+1}$, el cual es multiplicado por el vector $(1, x)$. Por lo tanto, el valor calculado en una neurona i perteneciente a la capa oculta viene dado por

$$a_i^{(1)} = g(\theta_i^{(1)^T} x)$$

siendo g la función de activación correspondiente, por lo general la función *sigmoide*,

$$g(z) = \frac{1}{1 + e^{-z}}$$

El valor calculado de la única neurona de salida se calcula de la siguiente forma

$$a_1^{(2)} = g(\theta_1^{(2)^T} a^{(1)})$$

donde $a^{(1)} = (1, a_1^{(1)}, \ldots, a_p^{(1)})$.

De forma que la salida de las neuronas de la capa final viene representado por $h_\Theta(x)$,

$$h_\Theta(x) = g(\theta_1^{(2)^T} a^{(1)}) = \frac{1}{1 + e^{-\theta_1^{(2)^T} a^{(1)}}}$$

una fórmula que incluye, como parámetros, los vectores $\theta_1^{(1)}, \ldots, \theta_p^{(1)}$ y $\theta_1^{(2)}$, que de forma general llamamos Θ.

Nótese que tanto la capa de entrada como las capas ocultas poseen un nodo extra llamado unidad de sesgo, con valor igual a 1. Este nodo permite que el umbral de decisión no tenga que pasar siempre por el origen.

La función de coste de una red neuronal con una única capa oculta calculada sobre un conjunto de entrenamiento $(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})$ donde $x^{(i)} \in \mathbb{R}^n, y^{(i)} \in \{0, 1\}$, se evalúa de la siguiente forma,

$$J(\Theta) = -\frac{1}{m} \left[\sum_{i=1}^{m} y^{(i)} \log h_\Theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\Theta(x^{(i)})) \right]$$

De forma adicional, a este sumatorio se puede añadir un parámetro de regularización λ que modifique la función de coste anterior, penalizando los pesos con un valor más alto con el objetivo de evitar el sobreajuste, descrito en la sección 3.6. Por lo tanto, si λ es demasiado grande todos los pesos se encontrarán en una escala muy baja haciendo que se generalice en exceso sobre los datos de entrenamiento. Por el contrario, un valor pequeño de λ podría no corregir el problema de sobreajuste.

El objetivo es encontrar los parámetros para que se minimice $J(\Theta)$. Con el fin de obtener el mejor redimiento posible de la red neuronal, los pesos de la red han de ser ajustados en base a la diferencia obtenida del valor calculado por la red y el valor real para cada muestra con la que se entrene la red, para ello se hace uso del algoritmo de propagación hacia atrás o también llamado *backpropagation* [19].

Al tratarse de un problema de regresión en vez de clasificación, la estructura de la red neuronal puede sufrir variaciones. La principal modificación es el remplazo de la función de activación de la neurona de salida, cambiando la función *sigmoide* por una función lineal dada por,

$$h_\Theta(x) = \theta_1^{(2)^T} a^{(1)}$$

Es posible seguir utilizando la función *sigmoide* si los datos son normalizados, en vez de estandarizados, en un rango [0-1], como se describe en la sección 3.2. Esto es debido a que los valores proporcionados por la función *sigmoide* son también aproximadamente [0-1] en su rango de salida.

3.3.1.1. Paquete nnet

Para la utilización de las redes neuronales en R se ha optado por el uso del paquete *nnet* [20] en detrimento del paquete *neuralnet* [21], el cual había sido pensado para ser utilizado en un primer momento. Dicho cambio se produce por la opción que proporciona *nnet* de utilizar la función *tune* del paquete *e*1071, la cual permite configurar diferentes tipos de arquitecturas de las redes neuronales en cuanto a número de neuronas se refiere y hacerlo usando validación cruzada *k-folds* con k=10, descrito en la sección 3.7. Esto hace posible seleccionar la configuración de la red, y por tanto el modelo, que proporcione un menor RMSE (Root Mean Square Error), descrito en la sección 3.5. La principal desventaja de este paquete es la imposibilidad de tener múltiples capas, por lo que todos los modelos son generados con una única capa oculta, a pesar de que la mayoría de los problemas tratados con redes neuronales no requieren más de una capa oculta al aumentar el tiempo de computo de forma considerable sin obtener mejoras significativas.

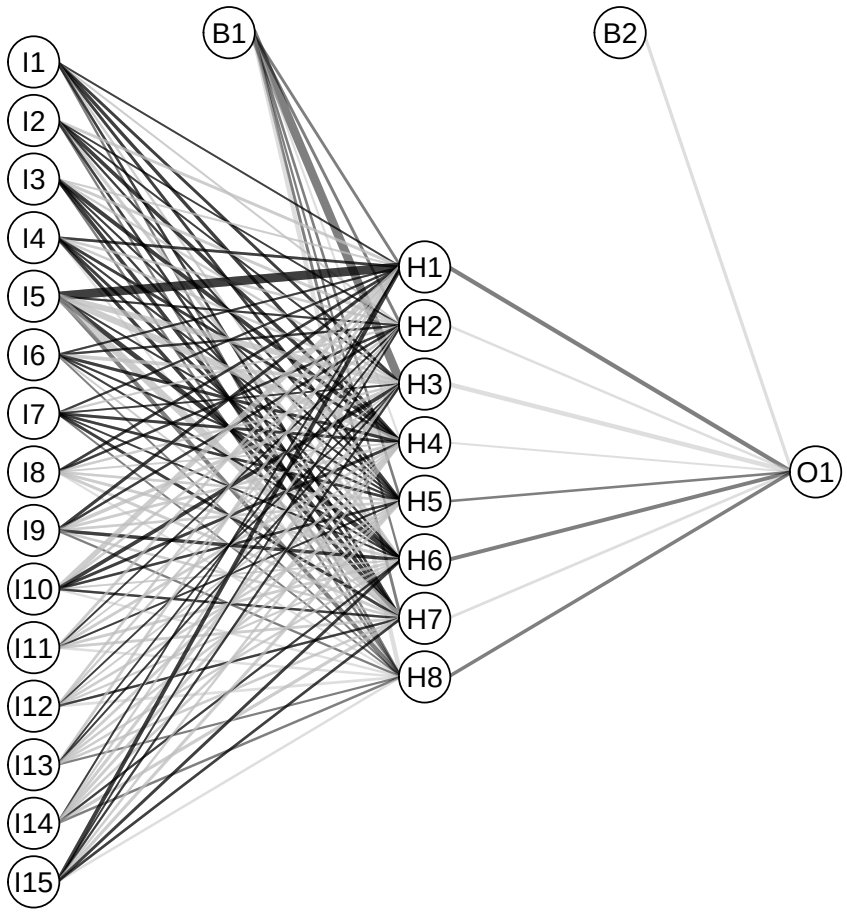

Figura 4: Red neuronal entrenada con *nnet*

La Figura 4 muestra la arquitectura de una red neuronal ya entrenada a diferencia de la Figura 3, con 3 señales distintas: temperatura, humedad y viento. Para cada una de las señales la red posee 5 nodos de entrada, que corresponden con los últimos 5 valores de la señal respecto al valor siguiente de la señal a predecir. Quedando $[I_1 - I_5]$ como las entradas de la señal de temperatura, $[I_6 - I_{10}]$ para la humedad y $[I_{11} - I_{15}]$ para el viento, mientras O_1 es el siguiente valor de la señal, es decir, el valor perteneciente al atributo objetivo, en este caso corresponde a la señal de la temperatura. La diferencia de color entre las distintas conexiones de las neuronas corresponde con el valor de los pesos generados, una línea más marcada indica un mayor peso en esa conexión. Por ejemplo, la conexión cuyo peso es mayor corresponde a I_5 con H_1, el último valor de la temperatura con la primera neurona de la capa oculta.

3.3.2. Máquinas de soporte vectorial

Las máquinas de soporte vectorial o SVM [22] es un modelo de aprendizaje automático cuya idea inicial fue introducida a comienzos de los años 90 [23]. Dicho algoritmo tiene por objetivo el tratar de maximizar la distancia de separación entre los puntos pertenecientes a clases diferentes introduciendo un margen de separación. Para lograrlo, se trata de minimizar la función de coste sobre los parámetros θ,

$$\min_{\theta} C\left[\sum_{i=1}^{m} y^{(i)} cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{j=1}^{n}\theta_j^2$$

siendo $cost_1$, $cost_0$ las funciones,

$$cost_1(z) = \begin{cases} 1-z & \text{si } z \leq 1 \\ 0 & \text{si } z > 1 \end{cases}$$

$$cost_0(z) = \begin{cases} 1+z & \text{si } z \geq -1 \\ 0 & \text{si } z < -1 \end{cases}$$

mientras que C es el parámetro que permite definir en qué medida se ajustará el umbral de decisión sobre los datos de entrenamiento. Un valor alto de C provoca que el umbral se ajuste mucho a los datos, lo que puede ocasionar un problema de sobreajuste, descrito en la sección 3.6, al introducir una alta varianza y un bajo sesgo. Por lo tanto, un valor alto de este parámetro hace que la minimización de la función de coste se centre sobre,

$$\min_{\theta} \frac{1}{2}\sum_{j=1}^{n}\theta_j^2$$

bajo las restricciones:

$$\theta^T x^{(i)} \geq 1, \quad si \quad y^{(i)} = 1$$
$$\theta^T x^{(i)} \leq -1, \quad si \quad y^{(i)} = 0$$

Por el contrario, un valor bajo de C hace que el margen de separación sea mayor, haciendo que datos relevantes puedan no ser tomados en cuenta, obteniendo una varianza baja y un sesgo alto.

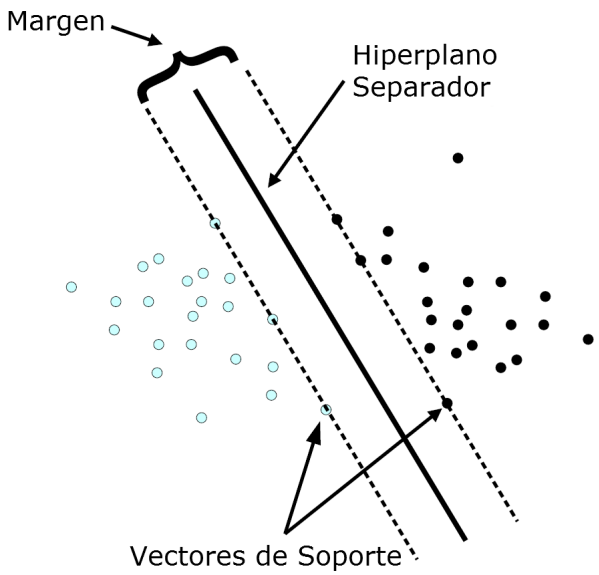

Figura 5: Umbral de decisión [24]

La Figura 5 muestra el umbral de decisión que separa dos clases que son linealmente separables, para ello maximiza la distancia entre ellas a través de las muestras más lejanas, llamados vectores de soporte.

En caso de tratarse de datos que no permiten su separación de forma lineal, se introduce una función llamada *kernel*. Esta función permite proyectar los datos a un espacio de mayor dimensión, permitiendo ser utilizada a modo de función de similaridad entre x y z, siendo z un punto de referencia. Uno de los *kernel* más utilizados es el *gaussiano*, el cual viene dado por,

$$K(x, z) = exp\left(-\frac{\|x - z\|^2}{2\sigma^2}\right)$$

Si el valor resultante es cercano a 1, significa que x y z guardan una alta similitud, mientras que un valor cercano a 0 indica lo contrario. De cara a utilizarlo con la máquina de soporte vectorial, se define $f^{(i)} \in \mathbb{R}^{m+1}$ para cada muestra i del entrenamiento de tamaño m, utilizando la función del *kernel* con $x^{(i)} \in \mathbb{R}^{n+1}$, siendo n el número de atributos de cada muestra i, y con los puntos de referencia elegidos de forma que $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \ldots, l^{(m)} = x^{(m)}$.

$$\begin{aligned}
&f_0^{(i)} = 1 \\
&f_1^{(i)} = K(x^{(i)}, l^{(1)}) = K(x^{(i)}, x^{(1)}) \\
&\vdots \\
&f_i^{(i)} = K(x^{(i)}, l^{(i)}) = K(x^{(i)}, x^{(i)}) \\
&\vdots \\
&f_m^{(i)} = K(x^{(i)}, l^{(m)}) = K(x^{(i)}, x^{(m)})
\end{aligned}$$

Una vez calculado f para cada muestra i del entrenamiento se procede a minimizar la función de coste,

$$\min_{\theta} C \left[\sum_{i=1}^{m} y^{(i)} cost_1(\theta^T f^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T f^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^{m} \theta_j^2$$

Para la parte de predicción, calculando previamente el f correspondiente a la nueva muestra es posible obtener los valores de la siguiente forma,

$$y = \begin{cases} 1 & \text{si } \theta^T f \geq 0 \\ 0 & \text{si } \theta^T f < 0 \end{cases}$$

Para el problema planteado en la sección 3.1, la maquina de soporte vectorial se aplica a la regresión en vez de a la clasificación, recibiendo el nombre de SVR [25]. La principal diferencia con SVM reside en la función de optimización, la cual viene dada por,

$$\min_{\theta} C \sum_{i=1}^{m} (\xi_i + \hat{\xi}_i) + \frac{1}{2} \sum_{j=1}^{m} \theta_j^2$$

$$\text{bajo las condiciones} \begin{cases} y^{(i)} - \theta^T f^{(i)} \leq \epsilon + \xi_i \\ -y^{(i)} + \theta^T f^{(i)} \leq \epsilon + \hat{\xi}_i \\ \xi_i, \hat{\xi}_i \geq 0 \end{cases}$$

donde $\xi_i, \hat{\xi}_i$ son variables de holgura que evitan que el sistema en el cual esas variables son nulas no tenga solución.

La Figura 6 muestra la curva de la regresión con el denominado ϵ-*tube* proporcionado por el parámetro ϵ y representado con la franja roja. ϵ controla el tamaño de dicha franja, si su valor es cercano a 0 se puede producir sobreajuste, mientras que por el contrario, si se trata de un valor alto se permite demasiada tolerancia al error haciendo que los resultados no sean satisfactorios al no ajustarse lo suficiente a los datos. Por encima de ϵ-*tube* se tiene $\xi_i > 0$ y $\hat{\xi}_i = 0$. Del mismo modo, los puntos por debajo de ϵ-*tube* tienen $\xi_i = 0$ y $\hat{\xi}_i > 0$. Por último, los puntos que se encuentran dentro de ϵ-*tube* contienen un valor de $\xi_i = \hat{\xi}_i = 0$ [17].

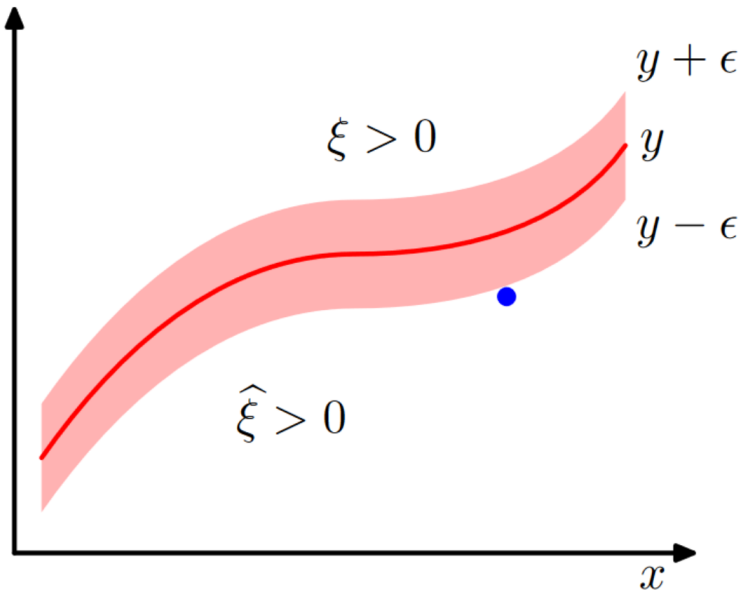

Figura 6: Curva de regresión con SVR [17]

3.3.2.1. Paquete e1071:svm

De cara a utilizar las máquinas de soporte vectorial en R, el paquete seleccionado es *e*1071 [26]. Este paquete contiene tanto la implementación de la máquina de soporte vectorial aplicada a la clasificación como a la regresión [24]. Así mismo, implementa la función *tune* que permite utilizar validación cruzada tanto para la máquina de soporte vectorial como para las redes neuronales en combinación con el paquete *nnet* descrito en la sección 3.3.1.1.

3.4. Autocorrelación

La autocorrelación se define como el coeficiente entre la covarianza de dos observaciones, y_t e y_{t-k}, y el producto de las varianzas individuales, siendo k el valor del desfase, también denominado *lag* [27],

$$\rho(k) = \frac{\text{cov}(y_t, y_{t-k})}{\sigma(y_t) \cdot \sigma(y_{t-k})} = \frac{\mathbf{E}[(y_t - \mu)(y_{t-k} - \mu)]}{\mathbf{E}[(y_t - \mu)^2]}$$

siendo σ la varianza, $\mathbf{E}$ la esperanza matemática y μ la media del conjunto de datos. Los valores posibles se encuentran en un intervalo $[-1, 1]$, siendo una función simétrica sobre el origen por lo que $p(k) = p(-k)$. De forma opcional se puede representar únicamente la parte positiva debido a su simetría, por lo que la parte negativa no proporciona información adicional [28].

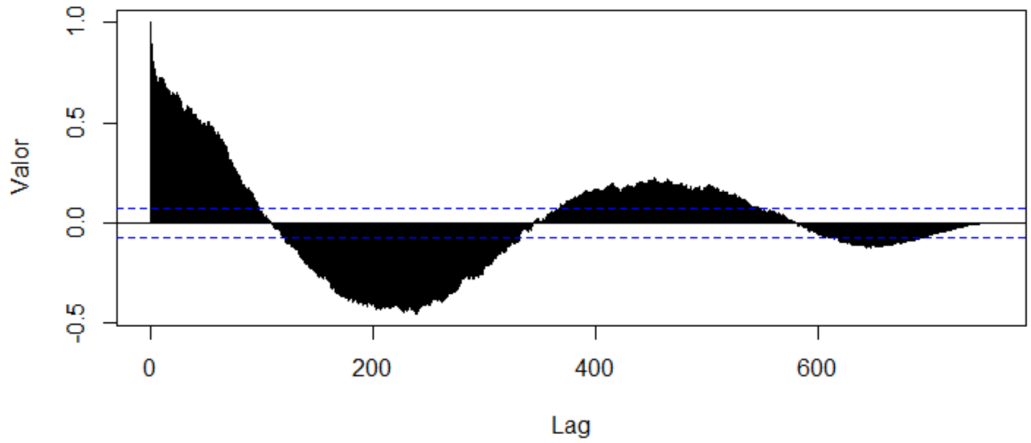

Figura 7: Autocorrelación de una señal en R

La Figura 7 muestra la autocorrelación perteneciente a una única señal de temperatura con 758 valores separados por 1 hora. El intervalo de confianza a partir del cual se consideran relevantes los valores de k viene dado por la línea discontinua azul, la cual proporciona un intervalo de confianza del 95 % [29] dado por,

$$r(95\,\%) = \frac{-1 \pm 1{,}96\sqrt{N-2}}{N-1}$$

donde N es el número de muestras utilizadas.

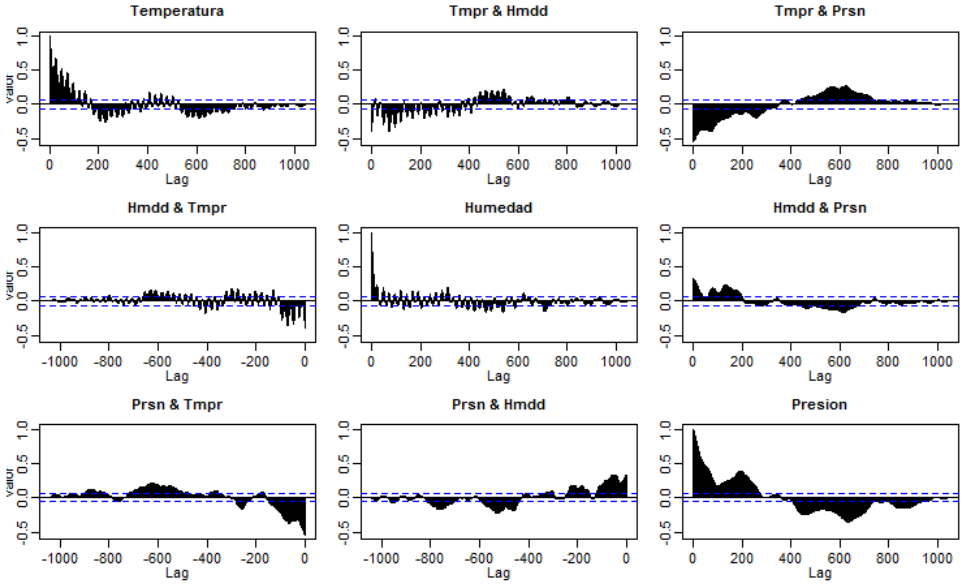

Figura 8: Autocorrelación de varias señales en R

La Figura 8 muestra la correlación cruzada perteneciente a las señales de temperatura, humedad y presión, todas ellas con valores separados por 1 hora, teniendo un total de 1043 valores cada una de ellas.

De cara a utilizar la autocorrelación en R, se hace uso del paquete acf, el cual se encuentra por defecto en R. Proporcionando el conjunto de datos deseados, acf devuelve los valores para los diferentes lag, así como una gráfica como las presentadas en las dos figuras anteriores.

3.5. RMSE (Root Mean Square Error)

Al tratarse de un problema de regresión, la manera de obtener una evaluación del modelo generado se hace en base a una medida denominada RMSE (Root Mean Square Error) [30], la cual es dada por,

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2}$$

siendo n el número de muestras evaluadas y e_i el error generado por la muestra i. Dicho error es calculado comparando el valor $\tilde{y}_i$ obtenido por el algoritmo, con el valor real y_i, siendo $e_i = \tilde{y}_i - y_i$.

3.6. Problema de sobreajuste

Uno de los problemas más frecuentes a la hora de utilizar algoritmos de aprendizaje automático, tanto para regresión como para clasificación, es la aparición de sobreajuste o también denominado $overfitting$ [17]. Este problema surge cuando en el proceso de entrenamiento, los datos disponibles se ajustan en exceso a la función generada por el modelo, provocando con la posterior llegada de nuevos datos una predicción de baja calidad.

Para solucionar este problema, una opción es incrementar el número de datos disponibles para el entrenamiento, haciendo que el algoritmo sea capaz de obtener una mayor flexibilidad de los mismos a la hora de realizar las predicciones. También es posible reducir el número de atributos de las muestras, utilizando únicamente los atributos más relevantes para el problema. Por último, otra opción consiste en utilizar parámetros de regularización, los cuales penalizan la función de optimización haciendo que la complejidad del modelo varíe.

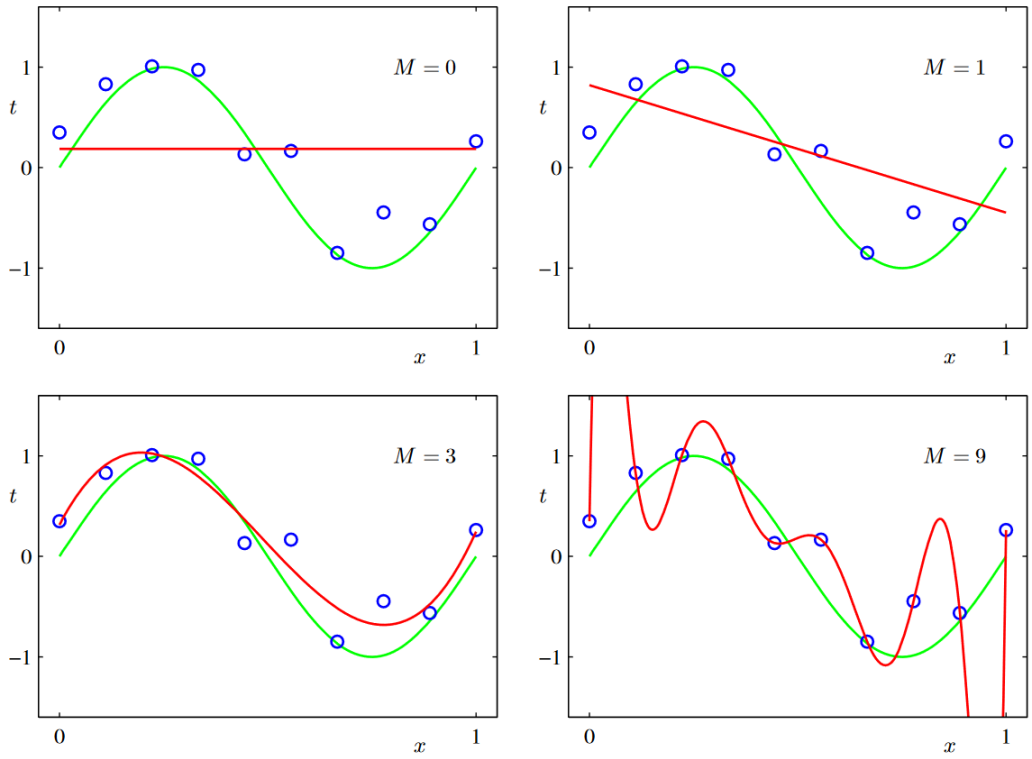

Figura 9: Polinomios con diferentes grados [17]

La Figura 9 muestra un ejemplo de diferentes polinomios representados con la línea roja en función del valor de M, el cual define el grado del polinomio. Por su parte, la línea de color verde representa la función $\sin(2\pi x)$, por lo que el objetivo es predecir el valor de t en función de un nuevo valor de x sin tener conocimiento de la línea verde, usando únicamente la información proporcionada por las muestras de color azul. Al aumentar el grado del polinomio se aprecia como la función de color rojo tiende a ajustarse cada vez más al grupo de puntos representados, llegando a ocurrir sobreajuste en los datos con $M = 9$.

3.7. Validación cruzada

La validación cruzada, y más concretamente, la validación cruzada mediante *k-folds* [31, 32] consiste en la repetición de la combinación entrenamiento-test varias veces con el fin de obtener una idea más fiel de la precisión del modelo generado. Para ello, el conjunto de datos D se divide en k subconjuntos o $folds$ $D_1, D_2, \ldots, D_k$ del mismo tamaño. El modelo L es entrenado y testeado k veces. Siendo $t \in \{1, 2, \ldots, k\}$ se testea sobre D_t y se entrena previamente sobre $\{D \backslash D_t\}$. Cada iteración t devuelve el error del modelo de la forma $E_t = L_t(D_t)$, siendo $L_t(D_t)$ el testeo del modelo L_t sobre el conjunto de datos D_t. El valor resultante es sumado al del resto de iteraciones y dividido por k al final, quedando el error de la validación cruzada como,

$$E_{vc} = \frac{1}{k} \sum_{t=1}^{k} E_t$$

Por norma general, y para la mayoría de las aplicaciones, se toma $k = 10$ [32], por lo que se realiza la validación cruzada mediante 10-*folds* tanto para las redes neuronales como para las máquinas de soporte vectorial.

3.8. Ajuste de hiperparámetros

El ajuste de hiperparámetros o también conocido como *Grid Search* [33] es una técnica que permite encontrar los parámetros que mejores resultados proporcionan sobre el algoritmo de aprendizaje automático que se desee aplicar. Para obtener dichos parámetros, se realiza una búsqueda sobre un espacio de soluciones en función del total de combinaciones posibles aportadas por los propios parámetros. Encontrada la combinación de parámetros que proporciona un menor RMSE, descrito en la sección 3.5, se dispone de la mejor configuración posible para el entrenamiento del modelo. Una forma de potenciar los efectos de esta técnica es usarla en combinación con la validación cruzada, descrita en la sección 3.7, para reducir el riesgo de producirse sobreajuste en el entrenamiento del modelo.

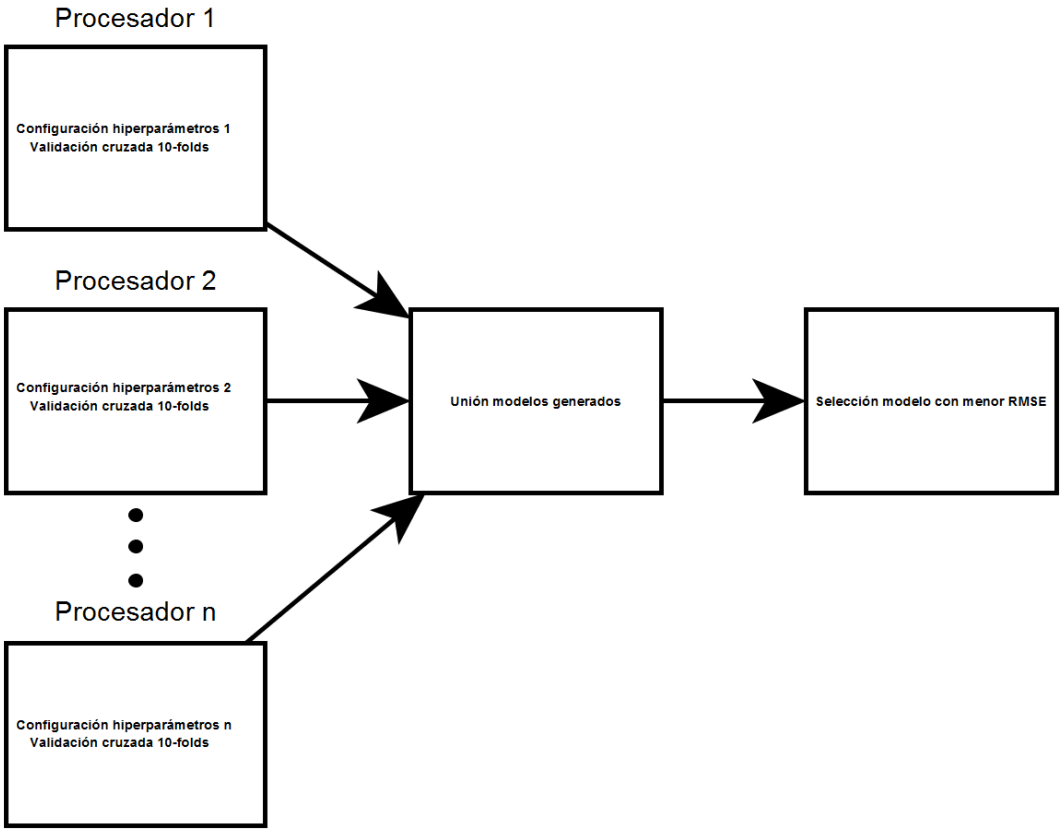

Figura 10: Ajuste de hiperparámetros

La Figura 10 muestra el flujo que se sigue hasta obtener el modelo que se utilizará para realizar las predicciones. En la primera fase, cada uno de los procesadores se encarga de generar modelos utilizando parte de los hiperparámetros que le han sido asignados, para aplicar validación cruzada 10-*folds* en cada uno de los modelos. Una vez todos los procesadores han terminado con sus respectivos modelos, estos se

unifican para posteriormente seleccionar el que tenga un menor RMSE.

En R es posible reproducir esta técnica haciendo uso del paquete *doParallel* [34] junto con la función *tune* ya comentada en secciones anteriores. El primero de ellos contiene una función denominada $foreach$, la cual se encarga de la ejecución simultánea en múltiples cores de las distintas combinaciones posibles de los parámetros. Esto hace que el tiempo de ejecución se reduzca considerablemente, y a su vez proporciona un índice de los valores de uno de los parámetros. Dicho índice será distribuido entre los procesadores de tal forma que cada uno de ellos trabaje sobre un rango concreto de valores. La función *tune* es la encargada de la otra parte del proceso, realizando la validación cruzada y teniendo otro índice con los valores de otro de los parámetros, el cual también será distribuido entre los procesadores en combinación con el primer índice. Hay que tener en cuenta que esta es la forma de realizarlo cuando solo es necesario obtener los mejores valores para dos parámetros, en caso de ser necesario optimizar más parámetros se utilizaría $foreach$ de forma anidada con la función *tune* encontrándose en el nivel más bajo de todos.

3.9. Postprocesamiento de los datos

De la misma forma que los datos son preprocesados a través de la normalización o de la estandarización antes de proporcionárselos al algoritmo correspondiente, ha de existir un proceso inverso. De esta manera, una vez realizada la predicción, es llevado a cabo el postprocesamiento de los datos. Es posible volver a una escala de valores perteneciente a la señal sobre la que se realiza la predicción. Para el caso de la normalización esto es posible dada la siguiente ecuación,

$$x = \tilde{x}(max - min) + min$$

teniendo en cuenta que max y min han de ser los valores máximo y mínimo respectivamente de los datos con los que se ha entrenado el algoritmo que corresponda, es decir, los datos de entrenamiento.

En el caso de la estandarización la ecuación utilizada viene dada por,

$$x = (\tilde{x} \cdot \sigma) + \mu$$

en donde tanto σ como μ son los valores utilizados para el entrenamiento del modelo.

4. Integración con IDbox

La integración con el producto IDbox se produce a través del desarrollo de la lógica de una serie de componentes software. Los componentes tienen por objetivo poder integrarse en un flujo de trabajo del BPM a través de una visualización basada en cajas conectadas. Un componente puede disponer de entradas procedentes de otros componentes, y salidas, que tienen la posibilidad de conectarse a otros componentes distintos en caso de ser necesario.

Cada uno de los distintos componentes se rige por una lógica implementada en C#, junto con la carga de los scripts de R haciendo uso de la librería R.NET. Para ello, se hace uso de dos proyectos diferentes, uno correspondiente al BPM y otro encargado de ejecutar el código de R, denominado NetR.

4.1. Proyecto BPM (Business Process Management)

Es el proyecto encargado de leer los parámetros de los componentes y los valores de las puertas de entrada, en caso de existir. También tiene como función hacer las peticiones al servicio de datos históricos para almacenarlos en un DTO (Data Transfer Object) junto con los parámetros que sean requeridos desde la parte de R. Por último, realiza la llamada al proyecto NetR a través de la creación de un hilo y se mantiene a la espera de un resultado para posteriormente transmitirlo a través de sus puertas de salida en caso de ser requerido.

4.2. Proyecto NetR

Es el proyecto encargado de recibir el DTO y de cargar los scripts de R para su ejecución. Por último, devuelve el resultado al proyecto BPM y un estado indicando si todo se ejecutó con normalidad o hubo algún fallo durante el proceso.

4.3. Componentes

Cada uno de los componentes tiene una clase propia tanto en el proyecto BPM como en el NetR, así como una clase para el DTO dentro también de NetR. A continuación son descritos los componentes junto con los parámetros que son requeridos para cada uno de ellos.

4.3.1. Autocorrelación

Permite obtener el mayor valor de autocorrelación, descrito en la sección 3.4, para el conjunto de datos proporcionado. Dicho valor será usado como número de puntos anteriores en los componentes de generación del modelo. El valor es obtenido una vez se ha eliminado un tanto por ciento de los valores calculados para los diferentes desfases, desde el inicio con el desfase igual a 0. Esta eliminación es llevada a cabo para que el usuario pueda encontrar un valor de autocorrelación correspondiente a un periodo de tiempo más grande en los datos, si esta eliminación no fuese realizada el valor de autocorrelación siempre estaría cercano al 0, haciendo que para el modelo que se genera posteriormente el número de puntos anteriores fuese muy pequeño y

por lo tanto proporcionando pocos valores de predicción que se puedan acercar a los valores reales, es decir, orientado a muy corto plazo.

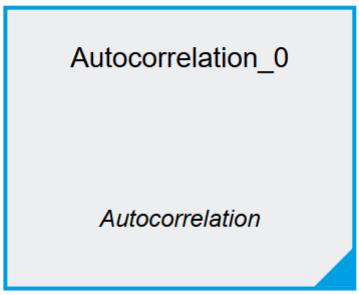

Figura 11: Componente autocorrelación

Toma en cuenta el siguiente parámetro introducido por el usuario:

- Porcentaje de eliminación (EliminationPercentage): número que establece que porcentaje de valores no serán tomados en cuenta a la hora de buscar el valor de autocorrelacion utilizado posteriormente como número de valores anteriores. Por ejemplo, el conjunto de datos mostrado en la Figura 13 (1008 valores) con un 15 % de eliminación supondría la eliminación de los valores de autocorrelación tal como se muestra en la Figura 12, donde se puede ver en rojo los índices de desfase eliminados y en verde los valores de los que saldrá el índice con mayor autocorrelación.

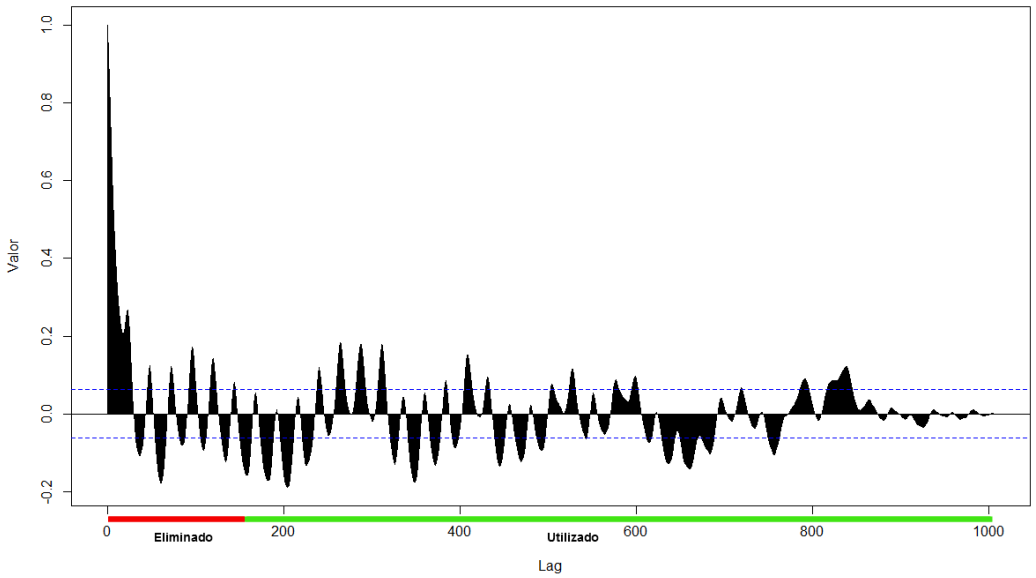

Figura 12: Eliminación en la autocorrelación

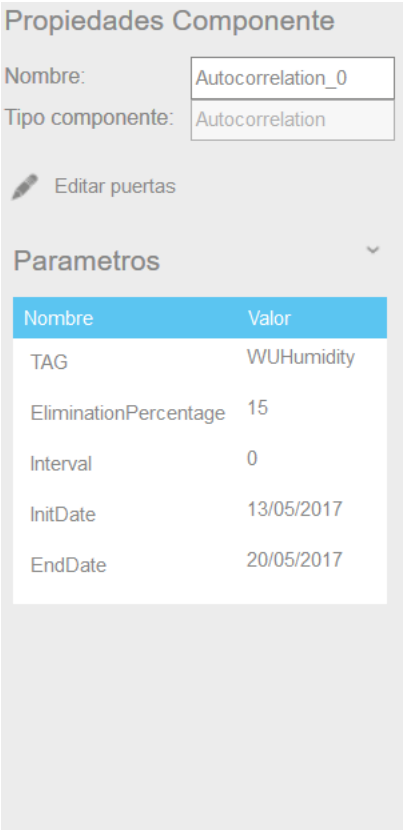

Figura 13: Propiedades autocorrelación

4.3.2. Redes Neuronales

Genera el modelo haciendo uso de las redes neuronales, descritas en la sección 3.3.1.

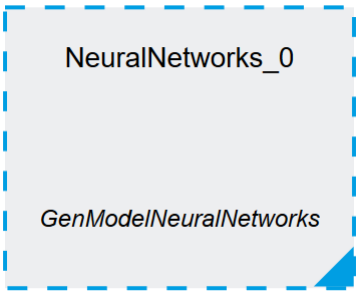

Figura 14: Componente redes neuronales

Toma en cuenta los siguientes parámetros introducidos por el usuario:

- Señal (TAG): señal o grupo de señales sobre las que se va a realizar el entrenamiento, en caso de ser varías son separadas por una barra vertical. La primera de las señales será tomada como la señal objetivo sobre la que posteriormente se realizará la predicción completa de los valores, mientras que para el resto de

señales se obtiene la predicción de valores intermedios necesarios que no son visibles para el usuario.

- Intervalo (Interval): resolución por la que se encuentran espaciados los datos entre ellos.
- Fecha de inicio (InitDate): fecha de comienzo de los datos históricos.
- Fecha de fin (EndDate): fecha de fin de los datos históricos.
- Neuronas (Neurons): rango de neuronas que serán utilizadas para el ajuste de hiperparámetros. Para ello se ha de especificar el número mínimo de neuronas junto con el número máximo, ambos valores separados por una barra vertical.
- Puntos anteriores (LastSamples): número entero calculado automáticamente con el componente de autocorrelación o introducido manualmente por el usuario. Especifica el número de puntos anteriores de cada señal utilizados para el entrenamiento con las muestras.
- String de conexión (ConnectionString): string de conexión a la base de datos
- Nombre del modelo (ModelName): nombre que se le asigna al modelo generado.

Figura 15: Propiedades redes neuronales

4.3.3. Máquinas de soporte vectorial

Genera el modelo haciendo uso de las máquinas de soporte vectorial, descritas en la sección 3.3.2.

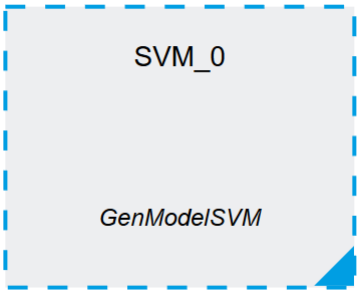

Figura 16: Componente SVM

Toma en cuenta los siguientes parámetros introducidos por el usuario:

- Señal (TAG): señal o grupo de señales sobre las que se va a realizar el entrenamiento, en caso de ser varías son separadas por una barra vertical. La primera de las señales será tomada como la señal objetivo sobre la que posteriormente se realizará la predicción completa de los valores, mientras que para el resto de señales se obtiene la predicción de valores intermedios necesarios que no son visibles para el usuario.
- Intervalo (Interval): resolución por la que se encuentran espaciados los datos entre ellos.
- Fecha de inicio (InitDate): fecha de comienzo de los datos históricos.
- Fecha de fin (EndDate): fecha de fin de los datos históricos.
- Épsilon (Epsilon): rango de valores reales que indican el mínimo, el máximo y el paso con que se ve incrementado el parámetro épsilon, todos ellos separados por barras verticales tal y como se muestra en la Figura 17.
- Coste (Cost): rango de valores reales que indican el mínimo, el máximo y el paso con que se ve incrementado el parámetro coste o también denominado C en la sección 3.3.2, todos ellos separados por barras verticales tal y como se muestra en la Figura 17.
- Puntos anteriores (LastSamples): número entero calculado automáticamente con el componente de autocorrelación o introducido manualmente por el usuario. Especifica el número de puntos anteriores de cada señal utilizados para el entrenamiento con las muestras.
- String de conexión (ConnectionString): string de conexión a la base de datos.
- Nombre del modelo (ModelName): nombre que se le asigna al modelo generado.

Figura 17: Propiedades SVM

4.3.4. Predicción

Realiza una predicción sobre el modelo que sea cargado desde la base de datos.

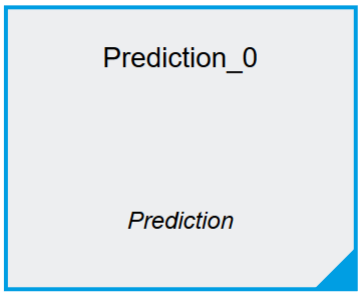

Figura 18: Componente predicción

Toma en cuenta los siguientes parámetros introducidos por el usuario:

- String de conexión (ConnectionString): string de conexión a la base de datos.
- Nombre del modelo (ModelName): nombre del modelo que se desea cargar desde base de datos.

- Señal (TAG): señal o grupo de señales utilizadas para realizar las predicciones, en caso de ser varías son separadas por una barra vertical. La primera de las señales será tomada como la señal objetivo sobre la que posteriormente se realizará la predicción completa de los valores, mientras que para el resto de señales se obtiene la predicción de valores intermedios necesarios que no son visibles para el usuario. El usuario tiene la posibilidad de introducir señales distintas a que las que se utilizaron para entrenar el modelo, aunque no se recomienda hacerlo.
- Puntos predicción (SamplesPredict): número de puntos siguientes que serán predichos para el modelo seleccionado.

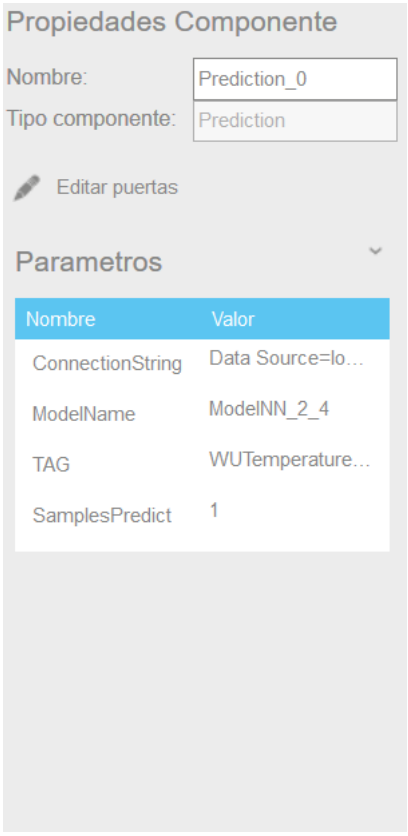

Figura 19: Propiedades predicción

4.4. Estructura del BPM

Todos los componentes disponibles para su uso en el BPM han de estar recogidos en una base de datos. Los parámetros, puertas de entrada y salida de cada uno de ellos, así como las conexiones entre diferentes componentes y los modelos generados también forman parte de la misma base de datos.

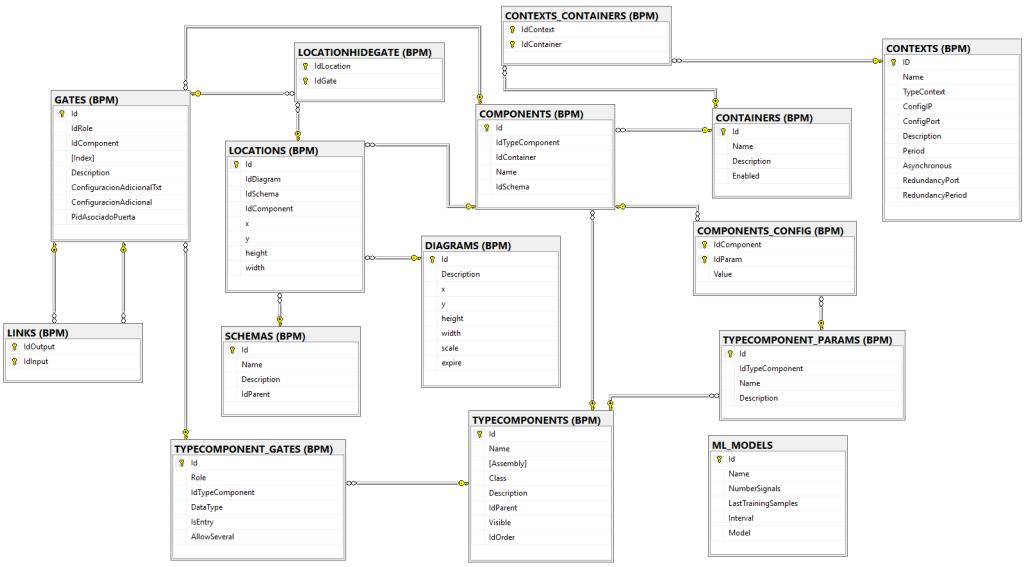

Figura 20: Diagrama de tablas del BPM (Business Process Management)

La Figura 20 muestra el diagrama de tablas correspondiente al BPM. A través del uso de varias de las tablas se han realizado las modificaciones necesarias en los datos, tanto para la creación de los componentes, así como para la creación de sus respectivos parámetros y conexiones entre ellos. A continuación se listan y describen únicamente las tablas utilizadas durante el proceso:

- *Contexts*: almacena los diferentes contextos de ejecución que pueden ser utilizados en el BPM, para el problema que se trata se ha creado un nuevo contexto denominado BPM_ML.
- *Contexts_Containers*: contiene las relaciones entre los contextos y los contenedores.
- *Containers*: recoge los distintos contenedores de componentes, agrupándolos de forma que los componentes del mismo tipo se visualicen de forma conjunta desde la web. Por lo tanto, los componentes de *Machine Learning* quedan congregados en un nuevo contenedor.
- *Typecomponents*: contiene los componentes creados para el flujo de trabajo correspondiente del BPM. Para la parte de *Machine Learning* se han incluido los componentes de autocorrelación, generación del modelo de SVM y de redes neuronales, y el componente de predicción.
- *Typecomponent_Params*: registra los parámetros correspondientes a los tipos de componentes almacenados en *Typecomponents*.
- *Typecomponent_Gates*: los diferentes tipos de puertas disponibles para su creación y posterior uso en los componentes quedan reflejados en esta tabla.
- *Components*: contiene las instancias de los componentes creados previamente en la tabla *Typecomponents*.

- *Components_Config*: almacena los parámetros de cada una de las instancias de los parámetros de los componentes, creados en la tabla denominada *Typecomponent_Params*.
- *Gates*: almacena las puertas disponibles para usar, ya sean puertas analógicas o digitales, tanto para entradas como para salidas.
- *Links*: contiene las conexiones existentes entre las puertas de las distintas cajas.
- *ML_Models*: tabla en donde se guardan los modelos a medida que son generados y desde donde son consultados para realizar las predicciones. Antes de poder ser almacenados han de pasar un proceso de serialización en el que se convierten en un flujo de bytes tal y como se muestra en la Figura 21, el proceso inverso se realiza cuando son cargados.

	Id	Name	NumberSignals	LastSamples	Interval	Model
1	30	ModeloPruebaNN_0	2	49	3600000	0x580A0000000200030302000203000000001300000006000000130000000...
2	31	ModeloPruebaSVR_0	2	49	3600000	0x580A0000000200030302000203000000001300000006000000130000000...
3	35	ModeloPruebaNN_0	2	49	3600000	0x580A0000000200030302000203000000001300000006000000130000000...
4	63	ModeloPruebaSVR_1	2	3	3600000	0x580A0000000200030203000203000000001300000006000000130000000...
5	1306	ModeloPruebaNN_RC	2	3	3600000	0x580A0000000200030203000203000000001300000006000000130000000...
6	1307	ModeloPruebaSVR_RC	2	25	3600000	0x580A0000000200030203000203000000001300000006000000130000000...
7	1314	ModeloPruebaNN_RC2	2	3	3600000	0x580A0000000200030203000203000000001300000006000000130000000...
8	1315	ModeloPruebaNN_Max	2	5	0	0x580A0000000200030203000203000000001300000006000000130000000...
9	1317	ModeloPruebaSVR_1H	2	5	3600000	0x580A0000000200030203000203000000001300000006000000130000000...
10	1318	ModeloPruebaSVR_Max1	2	5	0	0x580A0000000200030203000203000000001300000006000000130000000...
11	1320	ModelNN_3_4	3	4	0	0x580A0000000200030203000203000000001300000006000000130000000...
12	1321	ModelNN_2_4	2	4	0	0x580A0000000200030203000203000000001300000006000000130000000...
13	1395	ModelNN_2_4_15	2	4	900000	0x580A0000000200030203000203000000001300000006000000130000000...
14	1399	ModelNN_2_4_Max	2	4	0	0x580A0000000200030203000203000000001300000006000000130000000...
15	1400	ModelNN_3_4_Max	3	4	0	0x580A0000000200030203000203000000001300000006000000130000000...
16	1401	ModelNN_3_4_Max	3	4	0	0x580A0000000200030203000203000000001300000006000000130000000...
17	1402	ModelNN_1_4_Max	1	4	0	0x580A0000000200030203000203000000001300000006000000130000000...

Figura 21: Modelos almacenados en la base de datos

5. Caso práctico

Una forma de estimar el comportamiento de las predicciones obtenidas tanto para redes neuronales como para máquinas de soporte vectorial, es la realización de pruebas sobre conjuntos de datos de entrenamiento y posteriormente sobre datos de test, así como pruebas sobre el comportamiento de las mismas desde IDbox. Para ello, se a decidido hacer uso de señales climatológicas proporcionadas por CIC Consulting Informático.

5.1. Caso práctico R

A continuación se describen las especificaciones del equipo donde se han efectuado las pruebas, los parámetros de cada uno de los algoritmos, las características de los datos de entrenamiento y test, y por último se nombran las señales utilizadas (junto con su abreviatura).

- Especificaciones equipo:
  - Procesador: Intel(R) Core(TM) i7-4700HQ CPU @ 2.40GHz (hasta 3.20GHz)
  - Núcleos: 4 (8 cores lógicos)
  - Memoria: 16,0 GB
  - Sistema operativo: Windows 10 Pro 64 bits
- Algoritmos:
  - Redes Neuronales:
    - Neuronas: 1–10.
  - SVR:
    - Épsilon: 0.0–0.2 incrementado por 0.05
    - Coste: 2.0–4.0 incrementado por 1.0
- Datos entrenamiento:
  - Fechas: 12/12/2016 11:00:00 - 16/04/2017 18:00:00
  - Intervalo: 1 hora
  - Datos totales: 3000
- Datos test:
  - Fechas: 16/04/2017 19:00:00 - 10/05/2017 11:00:00
  - Intervalo: 1 hora
  - Datos totales: 568
- Señales:
  - Temperatura (T) - Objetivo

- Humedad (H)
- Presión (P)
- Radiación solar (RS)
- Punto rocío (PR)
- Velocidad viento (VV)

Señales		Redes Neuronales			SVR		
Entrenamiento	Objetivo	P=4	P=24	P=48	P=4	P=24	P=48
T	T	0,7886	0,7831	0,7663	0,7600	0,6478	0,5772
T, H	T	0,7359	0,7798	0,9491	0,7362	0,5143	0,4703
T, H, P	T	0,7306	0,8001	0,6604	0,7007	0,4782	0,4008
T, H, P, RS	T	0,6497	0,6746	0,6759	0,5317	0,4139	0,3120
T, H, P, RS, PR	T	0,6706	0,6432	0,5930	0,5753	0,3813	0,3123
T, H, P, RS, PR, VV	T	0,6424	0,8748	0,5727	0,5442	0,3386	0,2592

Tabla 5: RMSE - Datos de entrenamiento

La Tabla 5 recoge los resultados en términos de RMSE de las diferentes configuraciones utilizadas sobre los datos de entrenamiento, es decir, para la generación de los modelos.

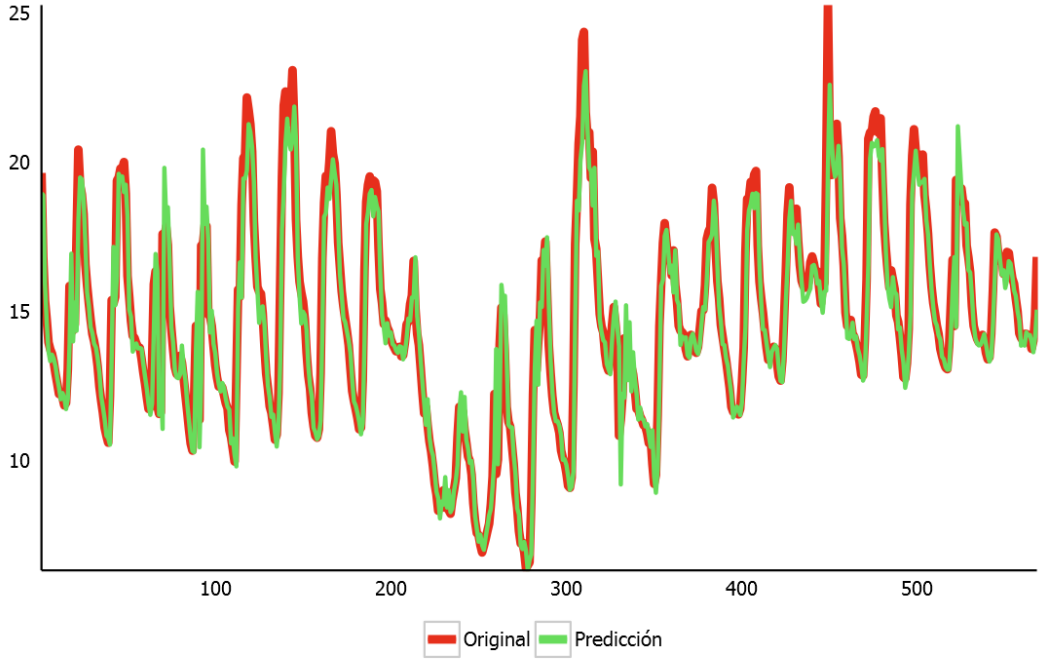

Figura 22: Comparación - Datos test originales vs predicciones

La Figura 22 muestra a modo de ejemplo los datos de test originales junto con la predicción realizada sobre los mismos datos en base al modelo generado mediante redes neuronales haciendo uso únicamente de la señal de temperatura y teniendo

en cuenta los últimos 4 valores para predecir el siguiente valor, haciendo que la predicción total sea de 568 valores.

Señales		Redes Neuronales			SVR		
Entrenamiento	Objetivo	P=4	P=24	P=48	P=4	P=24	P=48
T	T	1,0902	1,0356	1,0396	1,0835	1,1132	1,1340
T, H	T	1,0776	1,0290	1,2354	1,0977	1,3800	1,4761
T, H, P	T	1,0656	1,0920	1,2604	1,3331	1,5619	1,7546
T, H, P, RS	T	0,9837	1,1517	1,5960	1,2989	1,5502	1,7501
T, H, P, RS, PR	T	1,0497	1,1255	1,6540	1,2663	1,5886	1,9251
T, H, P, RS, PR, VV	T	1,0231	1,2872	2,0968	1,3507	1,7923	2,2047

Tabla 6: RMSE - Datos de test

La Tabla 6 recoge los resultados en términos de RMSE de las diferentes configuraciones utilizadas sobre los datos de test, es decir, para la generación de los modelos. También se encuentra recogido el RMSE generado por la Figura 22, en este caso con un valor de $1,0902$.

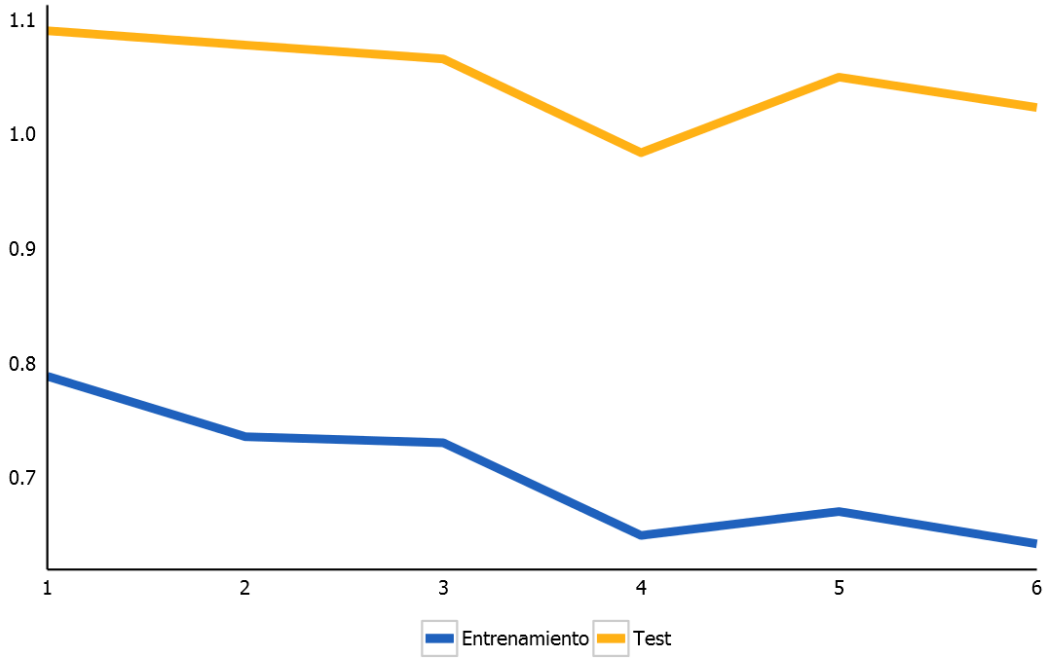

Figura 23: Comparación RMSE Redes Neuronales - Entrenamiento vs Test

La Figura 23 compara el error obtenido en la Tabla 5 para los datos de entrenamiento, con el error generado por los datos de test de la Tabla 6 para redes neuronales utilizando los últimos 4 valores. Como se ve en la gráfica, a medida que se aumenta el número de señales ambos errores tienden a disminuir. Dichas señales corresponden con las utilizadas durante el entrenamiento, es decir, en el eje x de la Figura 23 el valor 1 corresponde al error del modelo generado usando únicamente la señal de temperatura, 2 es el error del modelo de la temperatura junto con la

humedad, y así sucesivamente hasta llegar a las 6 señales recogidas en las Tablas 5 y 6.

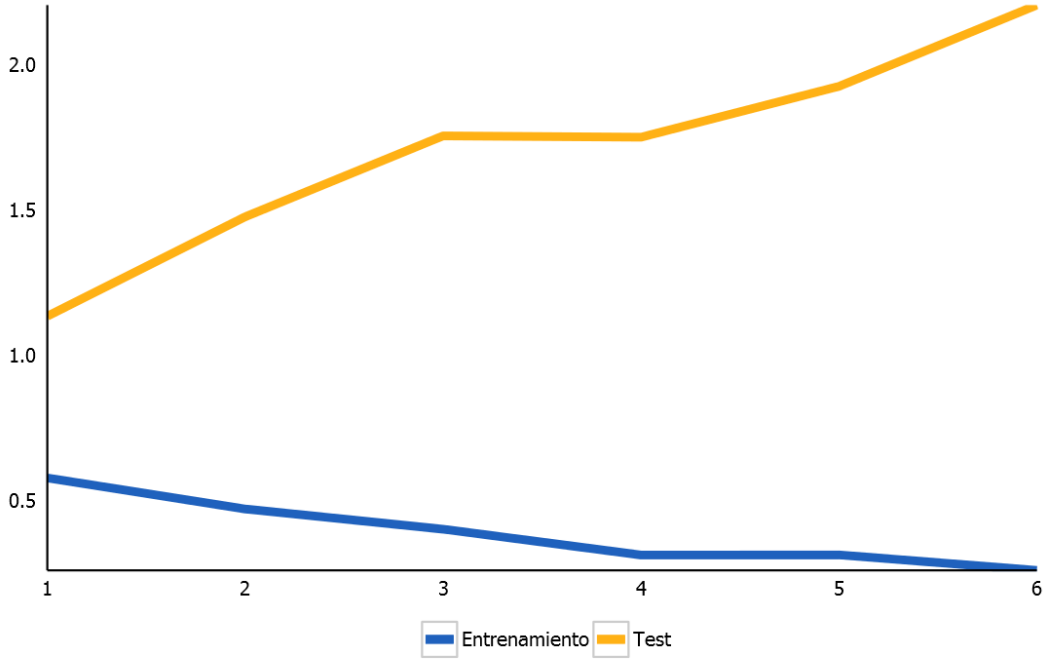

Figura 24: Comparación RMSE SVR - Entrenamiento vs Test

Por su parte, la Figura 24 muestra también la comparación del error, pero utilizando en este caso los últimos 48 valores para las máquinas de soporte vectorial. Mientras que el error para los valores de entrenamiento disminuye a medida que se introducen nuevas señales (aumento de la complejidad del modelo), para los valores de test se ve incrementado produciéndose el problema de sobreajuste descrito en la sección 9.

Señales		Redes Neuronales			SVR		
Entrenamiento	Objetivo	P=4	P=24	P=48	P=4	P=24	P=48
T	T	15,10	24,91	42,60	28,65	49,12	70,20
T, H	T	31,79	76,52	137,69	65,34	141,00	234,31
T, H, P	T	54,55	150,84	309,71	113,43	263,98	449,04
T, H, P, RS	T	83,40	257,59	597,65	179,88	428,09	766,77
T, H, P, RS, PR	T	113,01	410,65	1069,99	230,35	643,75	1170,30
T, H, P, RS, PR, VV	T	151,80	621,36	1870,65	321,07	933,32	1756,53

Tabla 7: Tiempo ejecución - Generación de modelos

A través de la Tabla 7 se pueden observar los tiempos de ejecución medidos en segundos para las distintas configuraciones utilizadas durante las pruebas. Correspondiendo el tiempo de ejecución más bajo a la utilización de una señal con los últimos 4 valores para redes neuronales, y el tiempo más alto al uso de los últimos 48 valores correspondientes a 6 señales para redes neuronales también, con un tiempo superior a los 31 minutos (1870,65 segundos).

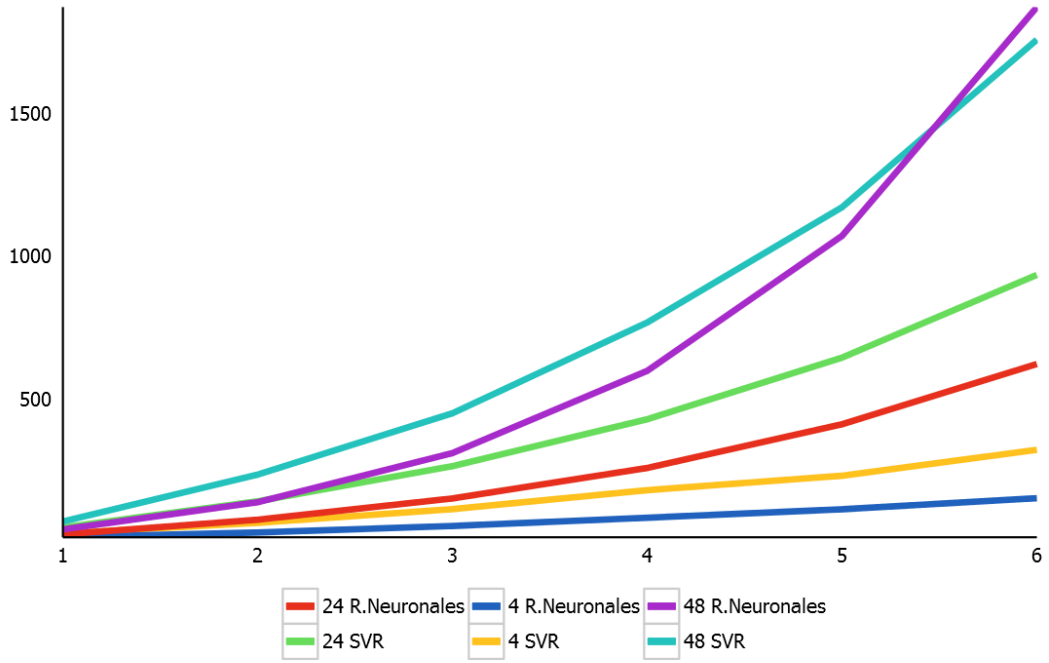

Figura 25: Comparación del tiempo de ejecución

La Figura 25 recoge la comparación del tiempo requerido, medido en segundos, para la generación de los modelos en función del número de señales utilizadas y graficando varios modelos con distinto número de últimos valores necesarios para la realización de las predicciones. Como se aprecia en dicha figura, un número mayor de último valores implica un incremento considerable en la tasa de crecimiento con respecto al uso de pocos valores, haciendo que sea recomendable generar modelos con pocos últimos valores, ya que como queda reflejado en la Tabla 6, $P = 4$ para redes neuronales es el único valor que no genera sobreajuste en las pruebas realizadas.

5.2. Caso práctico IDbox

Los pasos necesarios para generar de forma correcta un modelo y posteriormente obtener predicciones desde IDbox son descritos a continuación a través de la creación de un diagrama donde se integren todos los componentes necesarios:

Paso 1: Creación de un componente de Autocorrelación en caso de optar por obtener el número de puntos anteriores para la generación del modelo de forma automática, por lo que se trata de un paso opcional. A dicho componente se le pasa el porcentaje de datos que no se tendrán en cuenta a la hora de encontrar una periodicidad en los datos, como se describe en la sección 4.3.1.

Paso 2: Creación de uno de los dos componentes que genera modelos, ya sea el componente de tipo Redes neuronales o el de Máquinas de soporte vectorial, también existe la posibilidad de añadir ambos componentes tal y como se muestra en la Figura 26. Para cada uno de ellos se requieren parámetros comunes a ambos o parámetros específicos de cada uno de los algoritmos. En caso de ser creado

el componente de Redes neuronales el parámetro específico corresponde con el número de neuronas. Mientras que para la creación del componente de Máquinas de soporte vectorial los parámetros específicos corresponden con 'Épsilon' y 'Coste'. Los parámetros comunes a ambos algoritmos corresponden con el *TAG*, el intervalo, la fecha de inicio y de fin, el número puntos anteriores (el cual podrá ser previamente calculado con el componente de Autocorrelación o introducido de forma manual por el usuario), el string de conexión y el nombre del modelo.

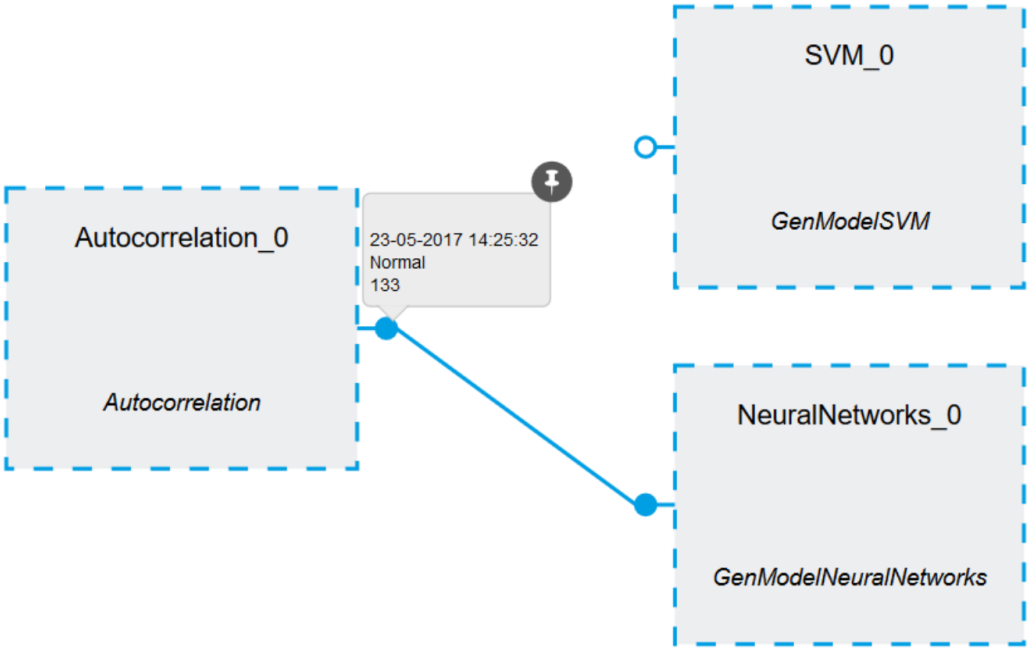

Figura 26: Componentes generación modelos

Paso 3: Una vez creados los componentes anteriores, el último paso es la creación del componente de Predicción. En este componente se selecciona un modelo generado previamente, junto con las señales deseadas, el string de conexión y el número de valores de predicción que se desean conocer desde el momento actual. Es importe tener en cuenta que los valores generados estarán a la misma frecuencia que los datos con los que se realizó el entrenamiento del modelo. Una forma de comparar los resultados de las predicciones con los valores de la señal real consiste en añadir un *TAG* en la salida de la caja de predicción, tal y como se aprecia en la Figura 27 a través del uso de la señal ML_Pruebas en donde se van almacenando los valores obtenidos. En la misma figura se puede ver como se ha añadido un componente de tipo *Timer* al diagrama, el cual tiene por objetivo generar cada x tiempo los valores de predicción mediante una expresión *Cron*. Los diagramas del BPM también dan la opción al usuario de ver los valores de salida y entrada de los componentes a través de pequeños rótulos como se muestra en las Figuras 26 y 27.

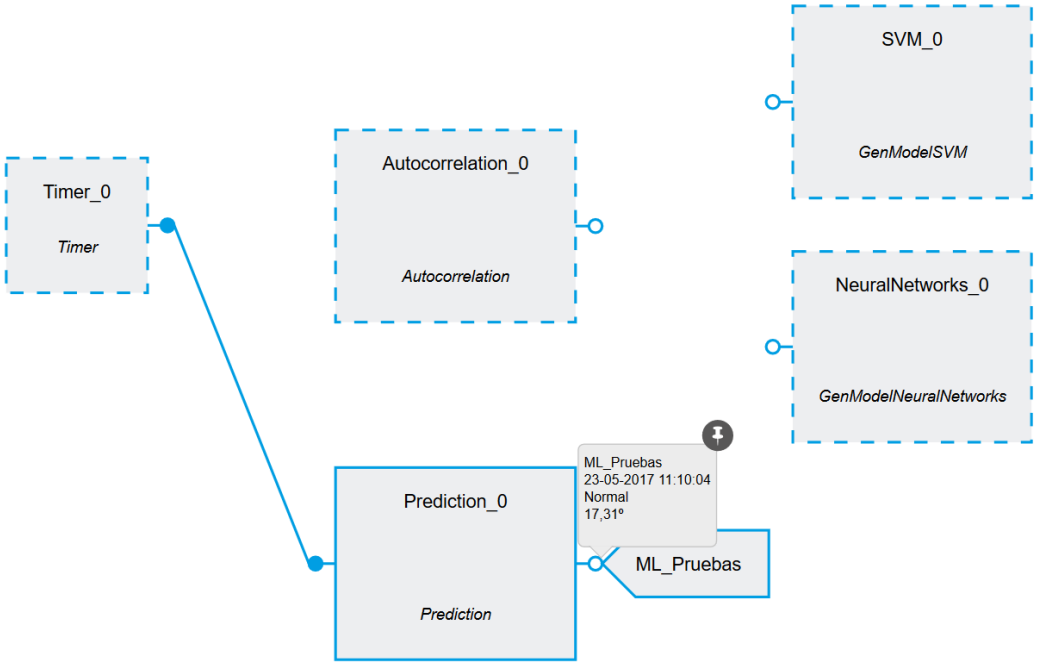

Figura 27: Predicción IDbox

Paso 4: Por último, tal y como muestra la Figura 28, se puede visualizar tanto la señal real (WUTemperature) como las predicciones obtenidas (ML_Pruebas) prediciendo cada vez un nuevo valor, lo que quiere decir que al únicamente obtenerse una predicción completa, se usan datos reales y no predicciones intermedias como podría ser necesario en caso de predecir más de un valor. Además se grafica la diferencia entre ambas señales, es decir, el error, apreciándose como el error medio es de $1,30$. Pasando dicho error a la métrica utilizada a lo largo de todo el proyecto se obtiene $RMSE = \sqrt{1,30} = 1,14$.

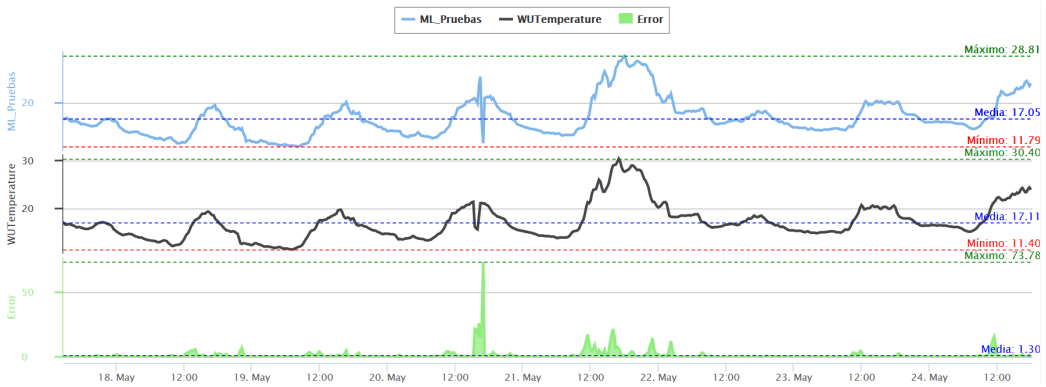

Figura 28: Comparación de las predicciones en IDbox

6. Conclusiones

El objetivo principal del proyecto consistía en el desarrollo de la lógica de una serie de componentes software que fuesen capaces de producir un flujo de trabajo en el que se obtuviesen una serie de predicciones al final del mismo haciendo uso de algoritmos de aprendizaje automático.

Cumplidos los objetivos descritos anteriormente, se pasa a valorar los resultados obtenidos para las pruebas realizadas tanto para la parte de R como para la de la integración final con el producto IDbox.

Los resultados en cuanto al RMSE obtenido en las predicciones se podrían definir como satisfactorios teniendo en cuenta las limitaciones del equipo en cuanto a memoria RAM, número de núcleos y que se ha hecho uso de señales climatológicas, las cuales suponen un grado de dificultad extra que sumar al problema tratado. Pudiéndose realizar pruebas en un equipo con unas prestaciones más altas, los resultados podrían ser mejorados al poder hacer uso de un mayor historial de datos para la generación de los modelo, es este punto el que me parece de mayor influencia de cara a obtener unas predicciones más acertadas, por encima de incrementar el número de señales o de aumentar el número de valores últimos utilizados. Aumentando considerablemente el histórico de datos se podrían solucionar problemas como el sobreajuste ocurrido en las pruebas con las máquinas de soporte vectorial.

La integración con el producto IDbox ha permitido visualizar los resultados del componente de predicciones almacenando los valores en una nueva señal, de forma que esta puede ser comparada con la señal real.

En todas las fases por las que ha pasado la realización del proyecto han surgido diferentes problemas y dificultades. Una de las primeras adversidades fue el hecho de tener que seleccionar los paquetes de R que mejor se adaptasen al problema tratado, para ello hubo que hacer pruebas no recogidas en la memoria con diferentes paquetes haciendo mayor hincapié en la selección del paquete usado para las redes neuronales. Según avanzaba el proyecto se llegó a la parte de la implementación de los scripts de R en C#, surgiendo problemas con el funcionamiento del BPM, ya que los componentes en vez de ejecutarse una vez se ejecutaban de forma repetida sin razón aparente. Otro de los problemas surgidos ha sido que el BPM no permite pasar más de un valor entre los componentes, haciendo que no pueda existir un componente donde se cargasen los históricos una única vez. La forma de solucionarlo ha sido cargando en cada uno de los componentes que interviniesen en el flujo de trabajo los históricos de forma individual. Por parte de la empresa se tiene previsto cambiar la forma de transferencia de datos entre los componentes para resolver este problema.

En cuanto a la parte personal, realizar el proyecto en una empresa me ha supuesto el poder trabajar de primera mano en un proyecto real y de largo plazo, y más aún sobre algo que me resulta bastante interesante como es el aprendizaje automático. A excepción de la libreria R.NET y del lenguaje de programación C# (aunque guarda cierta similitud con Java), tanto SQL como R han sido utilizados en asignaturas

durante la carrera, esto ha facilitado en parte el desarrollo del proyecto.

De cara a la mejora del proyecto realizado, se pueden listar los siguientes puntos clave que a mi entender podrían desarrollarse en el futuro para dar continuidad al proyecto y mejorar lo ya existente:

- Implementación de nuevos algoritmos de aprendizaje automático que puedan ser utilizados para problemas de regresión, como por ejemplo *Random Forest* [35].
- Creación de un componente que permita obtener un conjunto histórico de datos sin tener que estar cargando dicho conjunto en cada uno de los componentes en los que se utiliza. Para ello debería modificarse el BPM haciendo que se puedan pasar múltiples datos entre los componentes, ya que por el momento solo se puede transferir un único valor de uno a otro.
- Reemplazo en el ajuste de hiperparámetros del uso de *Grid Search* por algoritmos genéticos [36] que permitan obtener unos parámetros óptimos o cercanos a los óptimos en un menor tiempo.
- Generación de predicciones a la vez para diferentes señales, por el momento se puede entrenar con varias señales pero obteniendo sólo la predicción completa para una de ellas.
- Añadir la opción de enviar una notificación al usuario cuando finalice la generación del modelo, por ejemplo, vía email.

Referencias

[1] M. Rüßmann, M. Lorenz, P. Gerbert, M. Waldner, J. Justus, P. Engel, and M. Harnisch, "Industry 4.0: The future of productivity and growth in manufacturing industries," *Boston Consulting Group*, p. 14, 2015.

[2] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for industry 4.0-based manufacturing systems," *Manufacturing Letters*, vol. 3, pp. 18–23, 2015.

[3] R. C. Team, "R language definition," *Vienna, Austria: R foundation for statistical computing*, 2000.

[4] R. Ihaka and R. Gentleman, "R: a language for data analysis and graphics," *Journal of computational and graphical statistics*, vol. 5, no. 3, pp. 299–314, 1996.

[5] J. Liberty, *Programming C#: Building. NET Applications with C.* .ºReilly Media, Inc.", 2005.

[6] H. Halvorsen, "Structured query language," *Department of Electrical Engineering*, pp. 15–18, 2012.

[7] J.-M. P. Kosei Abe, "R.NET," *https://www.nuget.org/packages/R.NET.Community/*. Accedido: 13-03-2017.

[8] R. Team, "Rstudio: integrated development for R," *RStudio, Inc., Boston, MA http://www.rstudio.com, year=2015*.

[9] J. S. Racine, "RStudio: A platform-independent IDE for R and Sweave," *Journal of Applied Econometrics*, vol. 27, no. 1, pp. 167–172, 2012.

[10] J. Mayo, *Microsoft Visual Studio 2010: A Beginner's Guide*. McGraw-Hill, Inc., 2010.

[11] R. Dewson, "Sql server management studio," in *Beginning SQL Server for Developers*, pp. 25–42, Springer, 2015.

[12] W. W.-S. Wei, *Time series analysis*. Addison-Wesley publ Reading, 1994.

[13] M. Hibon and S. Makridakis, "ARMA models and the Box–Jenkins methodology," 1997.

[14] G. P. Zhang, "Time series forecasting using a hybrid arima and neural network model," *Neurocomputing*, vol. 50, pp. 159–175, 2003.

[15] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

[16] S. Aksoy and R. M. Haralick, "Feature normalization and likelihood-based similarity measures for image retrieval," *Pattern recognition letters*, vol. 22, no. 5, pp. 563–582, 2001.

[17] C. M. Bishop, "Pattern recognition," *Machine Learning*, vol. 128, pp. 1–58, 2006.

[18] H. B. Demuth, M. H. Beale, O. De Jess, and M. T. Hagan, *Neural network design*. Martin Hagan, 2014.

[19] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[20] B. Ripley, W. Venables, and M. B. Ripley, "Package 'nnet'," *R package version*, pp. 7–3, 2016.

[21] S. Fritsch, F. Guenther, and M. F. Guenther, "Package 'neuralnet'," 2016.

[22] A. Ng, "Cs229 lecture notes, part v: Support vector machines," *Technical report*, 2012.

[23] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144–152, ACM, 1992.

[24] D. Meyer and F. T. Wien, "Support vector machines," *The Interface to libsvm in package e1071*, 2015.

[25] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and computing*, vol. 14, no. 3, pp. 199–222, 2004.

[26] D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, F. Leisch, C.-C. Chang, C.-C. Lin, and M. D. Meyer, "Package 'e1071'," *The Comprehensive R Archive Network. Available at:*`https://cran.r-project.org/web/packages/e1071/e1071.pdf`*, year=2015*.

[27] R. M. Levich and R. C. Rizzo, "Alternative tests for time series dependence based on autocorrelation coefficients," *Working paper series-New York University Salomon Center S*, 1999.

[28] P. M. Broersen, *Automatic autocorrelation and spectral analysis*. Springer Science & Business Media, 2006.

[29] D. Meko, "GEOS 585A, Applied Time Series Analysis," 2005.

[30] T. Chai and R. R. Draxler, "Root mean square error (RMSE) or mean absolute error (MAE)?–Arguments against avoiding RMSE in the literature," *Geoscientific Model Development*, vol. 7, no. 3, pp. 1247–1250, 2014.

[31] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *IJCAI*, vol. 14, pp. 1137–1145, Stanford, CA, 1995.

[32] Z. Reitermanov, "Data splitting," in *WDS*, vol. 10, pp. 31–36, 2010.

[33] C.-W. Hsu, C.-C. Chang, C.-J. Lin, *et al.*, "A practical guide to support vector classification," 2003.

[34] R. Analytics and S. Weston, "doparallel: Foreach parallel adaptor for the parallel package," *R package version*, vol. 1, no. 8, 2014.

[35] A. Liaw and M. Wiener, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[36] J. Yang and V. Honavar, "Feature subset selection using a genetic algorithm," *IEEE Intelligent Systems and their Applications*, vol. 13, no. 2, pp. 44–49, 1998.

A. Anexo I: Dependencias paquetes R

- *e*0171
  - *e*0171
  - *class*
  - *MASS*
  - *lattice*
  - *cluster*
  - *mlbench*
  - *nnet*
  - *randomForest*
  - *rpart*
  - *SparseM*
  - *xtable*
  - *Matrix*
- *doParallel*
  - *doParallel*
  - *foreach*
  - *iterators*
  - *codetools*
  - *lattice*
  - *ggplot*2
  - *car*
  - *plyr*
  - *ModelMetrics*
  - *nlme*
  - *reshape*2
  - *MASS*
  - *mgcv*
  - *nnet*
  - *pbkrtest*
  - *quantreg*
  - *digest*
  - *gtable*
  - *scales*
  - *tibble*
  - *lazyeval*

- *Rcpp*
- *stringr*
- *Matrix*
- *lme4*
- *SparseM*
- *MatrixModels*
- *RColorBrewer*
- *dichromat*
- *munsell*
- *labeling*
- *stringi*
- *magrittr*
- *assertthat*
- *minqa*
- *nloptr*
- *RcppEigen*
- *colorspace*
- *caret*
- *mlbench*
- *rpart*

- *foreach*
  - *foreach*
  - *codetools*
  - *iterators*
  - *randomForest*
- *Metrics*
  - *Metrics*
- *nnet*
  - *nnet*
  - *MASS*
- *reshape*
  - *reshape*
  - *plyr*
  - *Rcpp*